UNIVERSITY OF OULU

TIETO- JA SÄHKÖTEKNIIKAN TIEDEKUNTA

Elia Päivärinta

Tietoturvan riskitason määrittäminen Android-laitteissa sovellusten ohjelmointirajapintaa käyttäen

Diplomityö
Tietotekniikan tutkinto-ohjelma
Maaliskuu 2020

Päivärinta E. (2020) Tietoturvan riskitason määrittäminen Android-laitteissa sovellusten ohjelmointirajapintaa käyttäen. Oulun yliopisto, Tietotekniikan tutkinto-ohjelma. Diplomityö, 69 s.

TIIVISTELMÄ

Android-laitteiden tietoturva muodostuu monesta osatekijästä, kuten näytön lukituksen käyttämisestä ja tallennustilan salaamisesta. Tämän työn tavoitteena on perehtyä Androidin tietoturvaan ja tehdä sovellus, jolla voidaan mitata tietoturvan riskitasoa Android-laitteissa.

Tässä työssä tehtiin Android-sovellus, joka mittaa erilaisia tietoturvaan liittyviä osatekijöitä Android-laitteesta käyttäen tavallisille sovelluksille tarjottua ohjelmointirajapintaa ja muodostaa käyttäjälle näin mitatuista osatekijöiden tilasta tietoturvan riskitason. Tällaisia osatekijöitä ovat esimerkiksi näytön lukituksen käyttäminen, tallennustilan salaus ja avoimien wifi-verkkojen käyttäminen. Tarkoituksena oli selvittää, onko ohjelmointirajapinnassa tarpeeksi liikkumavaraa, jotta sovelluksesta saisi tehtyä hyödyllisen. Tarkistettavien asioiden ulkopuolelle jätettiin laitteeseen asennetut muut sovellukset ja laitteen lokitietojen tarkastelut niiden laajuuden vuoksi ja työn rajoittamiseksi.

Sovellus laskee tietoturvan riskitason mittaamalla laitteesta 11 eri osatekijää ja painottamalla niitä osatekijäkohtaisilla painoarvoilla, jotka muodostettiin aiheutuvan tietoturvauhan todennäköisyydellä ja vaikutuksella. Näitä 11:tä osatekijää tarkastellaan tässä työssä tarkemmin tutkien niiden tärkeyttä ja vaikutusta tietoturvaan sekä kerrotaan, miten niiden tilaan voi vaikuttaa. Mitattujen osatekijöiden perusteella muodostettu riskitaso ei ole kaikenkattava, mutta antaa hyvän lähtökohdan sovelluksen jatkokehitykselle.

Sovelluksella laskettu riskitaso voidaan lähettää myös eteenpäin toiselle sovellukselle. Tämä toinen sovellus voi olla järjestelmätason sovellus, jolla on tavallista sovellusta laajemmat käyttöoikeudet ja joka pystyy näin tekemään riskitason muutosten perusteella muutoksia laitteen toimintaan ehkäisten mahdollisia tietoturvauhkia. Tässä työssä kehitetty sovellus toimii missä tahansa Android 6-9 käyttöjärjestelmää käyttävässä laitteessa.

Kehitetyssä sovelluksessa todettiin olevan potentiaalia ja sen jatkokehitystä mietitään tässä työssä muodostetun pohjan perusteella. Sovelluksesta olisi jatkossa mahdollista tehdä myös järjestelmätason sovellus, jolloin se saisi laajemmat oikeudet tarkastella laitteen toimintaa ja ominaisuuksia.

Avainsanat: Android, tietoturvariski, suojausominaisuus, eristäminen, salaus, ADB

Päivärinta E. (2020) Information security risk level assessment on Android-devices through application programming interface. University of Oulu, Degree Programme in Computer Science and Engineering. Master's Thesis, 69 p.

ABSTRACT

Information security on Android devices consists of many factors, such as use of screen lock and storage encryption. The aim of this master's thesis is to learn about Android information security and create an application that measures the information security risk level of Android devices.

For this thesis, an Android application was created that measures various factors in information security of Android devices, using the Android application programming interface, and calculates the risk level based on the measured factors. These factors include, for example, use of a screen lock, storage encryption, and use of open wifi networks. The purpose was to determine whether the application programming interface provides sufficient ability to develop a useful application. Excluded from the scope were other applications installed on the device as well as device log information, because of the broadness of work that would be required to analyze them and to limit the thesis.

The application calculates the risk level by measuring 11 different information security risk factors from the device and weighting them individually based on both the estimated probability and impact of the information security threat each risk factor represents. Each of these 11 factors is also explored in more depth in this thesis in terms of their importance and impact on information security, and methods of reducing the level of risk are presented. These measured risk factors do not provide a fully complete measurement of the device's security risks, but they create a good foundation for the further development of the application.

The risk level calculated by the application can also be sent to another application. This second application could be a system-level application, which has wider privileges than a normal application and thus could react to changes in risk level by changing the behavior of the device to avoid information security threats. The application developed in this thesis works on any device using Android 6-9 operating system.

The developed application was seen to have potential and the possibilities for its future development are explored based on the foundation made by the developed application. It would also be possible to develop a system-level application on this same concept, which would grant broader privileges to observe actions and attributes of a device.

Key words: Android, information security risk, security feature, isolation, encryption, ADB

SISÄLLYSLUETTELO

ALKULAUSE

Haluan kiittää Bittiumia mahdollisuudesta tehdä tämä diplomityö osana heidän projektiaan. Olen erityisen kiinnostunut Androidista ja tietoturvasta, joten diplomityö ja projekti, jonka ohessa se tehtiin, olivat itselleni mielekkäitä aiheita. Pystyin myös itse vaikuttamaan diplomityön kehityssuuntaan, mikä teki työstä haastavampaa, mutta opetti samalla itsenäisempään kehitystyöhön.

Erityisesti haluan kiittää Jouni Hiltusta Bittiumilta sekä professori Juha Röningiä ja tutkijatohtori Satu Tammista Oulun yliopistolta työn ohjaamisesta ja tarkastamisesta. Sain Jounilta todella arvokasta ohjausta työn eri vaiheissa. Kiitän myös Meira Maaninkaa työn kieliasun tarkastamisesta.

Oulussa, 11. maaliskuuta 2020

Elia Päivärinta

LYHENTEIDEN JA MERKKIEN SELITYKSET

ADB	Android Debug Bridge
AOSP	Android Open Source Project
API	Application Programming Interface
ASLR	Address Space Layout Randomization
CA	Certificate Authority
CFI	Control Flow Integrity
CIS	Center for Internet Security -järjestö
CVE	Common Vulnerabilities and Exposures
DAC	Discretionary Access Control
DNS	Domain Name System
DoS	Denial of Service
eMMC	embedded MultiMediaCard
FBE	File Based Encryption
FDE	Full Disk Encryption
FRP	Factory Reset Protection
GDPR	General Data Protection Regulation
GPS	Global Positioning System
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ISP	In-System Programming
JSON	JavaScript Object Notation
JTAG	Joint Test Action Group
LAP	Lower Address Part
LSB	Least Significant Bit
MAC	Mandatory Access Control
MITM	Man In The Middle
MSB	Most Significant Bit
NAP	Non-significant Address Part
NFC	Near Field Communication
OEM	Original Equipment Manufacturer
PHA	Potentially Harmful Application
TCP/IP	Transmission Control Protocol / Internet Protocol
TEE	Trusted Execution Environment
TLS	Transport Layer Security
UAP	Upper Address Part
UID	User Identifier
USB	Universal Serial Bus
XML	Extensible Markup Language

1. JOHDANTO

Älypuhelimista ja tableteista löytyy nykyään ominaisuuksia, joita mobiililaitteissa ei kymmenen vuotta sitten ollut. Niillä voi käyttää monipuolisesti internetiä, Bluetoothia ja muita verkko-ominaisuuksia, ja niille löytyy valtava määrä erilaisia sovelluksia peleistä hyötytyökaluihin. Mobiililaitteita käytetäänkin yhä enemmän toimintoihin, joissa tietoturvan tärkeys korostuu, kuten mobiiliverkkopankki, lähimaksaminen, erilaisiin asiointipalveluihin kirjautuminen ja henkilökohtaisten tietojen selaus, esimerkiksi terveydenhoitopalveluissa. Älylaitteen joutuminen vääriin käsiin tai sen tietojen urkinta voivat siis aiheuttaa yhä suurempaa vahinkoa käyttäjälle. Laitteen tietoturvallinen käyttäminen ja sen tietoturvasta huolehtiminen on hyvä opetella samaan aikaan kuin laitteen muukin käyttö. Tämä tarkoittaa esimerkiksi näytön lukituksen käyttämistä, kaksivaiheisen tunnistautumisen käyttöä sitä tarjoavissa palveluissa sekä tarpeeksi pitkiä ja hankalasti arvattavia salasanoja.

Mobiilikäyttöjärjestelmistä tällä hetkellä suosituin on Android. StatCounter-sivuston mukaan tammikuussa 2020 älypuhelimista ja tableteista 72,57 % käytti Androidia. Toiseksi suosituin mobiilikäyttöjärjestelmä oli iOS 26,52 % osuudella ja muut, muun muassa Windowsin mobiilikäyttöjärjestelmät ja KaiOS, jakavat lopun 0,91 % osuuden [1]. Android onkin tässä mielessä houkutteleva kohde hakkereille ja sen tietoturvaa koetellaan monelta suunnalta. Tästä syystä erityisesti Android-laitteiden käyttäjien olisi hyvä olla tietoisia siitä, miten laitteita käytetään turvallisesti. Android-käyttöjärjestelmää voi löytää nykyään monenlaisista laitteista, kuten älypuhelimista, tableteista, älykelloista, älytelevisioista, kameroista ja pelikonsoleista. Tämän työn osalta keskitytään kuitenkin älypuhelimiin ja tabletteihin, sillä ne voidaan nähdä yleisemmin henkilökohtaisessa kommunikoinnissa käytetyiksi laitteiksi. Jatkossa laitteista puhuttaessa viitataan siis niihin.

1.1. Työn tavoite ja rajaus

Tässä työssä on tarkoitus kehittää sovellus, joka mittaa Android-laitteen tietoturvan riskitasoa tarkistamalla siitä erilaisia tietoturvaan vaikuttavia osatekijöitä ja laskee niiden perusteella laitteen riskitason. Tällaisia osatekijöitä ovat esimerkiksi näytön lukitus ja avoimien wifi-verkkojen käyttö. Osatekijöiden tila esitetään käyttäjälle, joka voi tämän yleisnäkymän avulla tarkkailla laitteensa tietoturvatasoa ja tehdä siihen parannuksia. Sovelluksen tulee myös pystyä lähettämään riskitasotieto toiselle sovellukselle, joka voi olla esimerkiksi järjestelmätason sovellus. Toisen sovelluksen ollessa järjestelmätason sovellus se voi kohonneen riskitason perusteella tehdä kovennuksia laitteen tietoturvaan. Kehitetyn sovelluksen tulee pystyä toimimaan Android 6–9 -käyttöjärjestelmissä.

Sovelluksen ei ole tarkoitus olla loppuun asti viimeistelty kaupallinen sovellus, vaan sen tarkoituksena on enemmänkin selvittää, onko Androidin sovellusten ohjelmointirajapinnassa tarpeeksi liikkumavaraa, jotta tällaisesta sovelluksesta saisi tehtyä hyödyllisen. Samalla pohditaan, saisiko sovelluksesta enemmän hyötyä, jos sen saisi toteutettua järjestelmätason sovelluksena tavallisen, ei-järjestelmätason sovelluksen sijasta. Järjestelmätason sovelluksilla on laajemmat käyttöoikeudet kuin tavallisilla sovelluksilla, ja se pystyisi näin ollen tarkastelemaan laitetta ja sen tilaa syvemmältä.

Tehtävän työn rajoittamiseksi sovelluksen ei tule tutkia laitteeseen asennettuja muita sovelluksia tai niiden toimintaa. Muiden sovelluksien toiminnan tutkiminen olisi myös haastavaa, koska sovellukset ovat hiekkalaatikoituja (engl. *sandboxing*), eli ne on eristetty toisistaan [2]. Myös laitteiden lokitietojen tarkastelu jätetään tutkimisen ulkopuolelle. Sovellusten ja lokien tarkistaminen ja seuraaminen tietoturvauhkien varalta vaatisi mahdollisesti koneoppimisalgoritmin kehittämistä, sillä pelkkä yksittäisten indikaattoreiden etsiminen ja seulominen ei ole isossa mittakaavassa järkevää.

Sovelluksen kehittämisen lisäksi tehdään katsaus Androidin tietoturva-ominaisuuksiin ja siihen, mistä ne rakentuvat. Sovelluksessa mitattavia tietoturvan osatekijöitä taustoitetaan myös tarkemmin ja selitetään, miksi ne ovat olennaisia asioita tarkistaa.

1.2. Työn motivaatio

Älypuhelimet ja tabletit ovat yleensä todella tärkeitä laitteita käyttäjilleen niiden sisältämien tietojen sekä niiden mahdollistamien toimintojen takia. Mobiililaitteille tehdään ja optimoidaan monenlaisia palveluita, kuten verkkokauppoja ja -pankkeja, joita käytettiin aiemmin lähinnä tietokoneilla. Mobiililaitteita käytetäänkin nykyään enemmän internetin selaamiseen kuin tavallisia tietokoneita [3]. Näin niillä käsitellään yhä enemmän henkilökohtaista tietoa, kuten viestejä, sähköposteja, käyttäjätunnuksia, kuvia ja muistiinpanoja. Mobiililaitteita käytetään myös töissä työvälineinä ja niillä voidaan käsitellä tärkeitä ja arkaluonteisia asiakas- ja yritystietoja.

Tietojen menetys tai niiden varastaminen mobiililaitteilta tuovat siis entistä enemmän vahinkoa käyttäjilleen. Tietomurrot ja toiminnan keskeytyminen hyökkäyksen tai murron seurauksena voivat olla yrityksille vakavia iskuja ja maksaa niille suuria summia sekä aiheuttaa asiakasmenetyksiä. Yritysmaailmassa ollaankin entistä paremmin perillä siitä, että tietoturva on tärkeä osa niiden toimintaa maailmassa, jossa esineet ja ihmiset ovat yhä enemmän yhteydessä toisiinsa internetin ja muiden verkkojen välityksellä. Samalla rikolliset ovat entistä ovelampia ja kyvykkäämpiä käyttämään hyväkseen heikosti tietoturvastaan huolehtivia henkilöitä ja yrityksiä.

Tietoturva on käsitteenä laaja ja joskus hieman epäselvä käsite. Tässä työssä pyritäänkin hahmottamaan suurimman käytössä olevan mobiilikäyttöjärjestelmän tietoturvaa ja -ominaisuuksia sekä antamaan työkalu Android-laitteiden tietoturvan riskitason mittaamiseen sovelluksen muodossa. Tietoturvaan vaikuttavia osatekijöitä, joita sovelluksella mitataan, otetaan tarkempaan tarkasteluun ja selvitetään, miksi niiden tila olisi hyvä tarkistaa ja korjata Android-laitteista.

1.3. Kontribuutio

Työssä tehty sovellus tehtiin Bittiumin projektille selvittämään tämäntyylisen sovelluksen potentiaalia ja kehittämismahdollisuuksia. Sovellukseen liittyvistä muista osista ja sovelluksista kerrotaan vain sen verran, mikä on tämän työn kannalta oleellista. Androidin tietoturvasta tehty taustoitus ja tietoturvan osatekijöiden tarkempi katsaus auttoivat myös sovelluksen jatkokehityksessä.

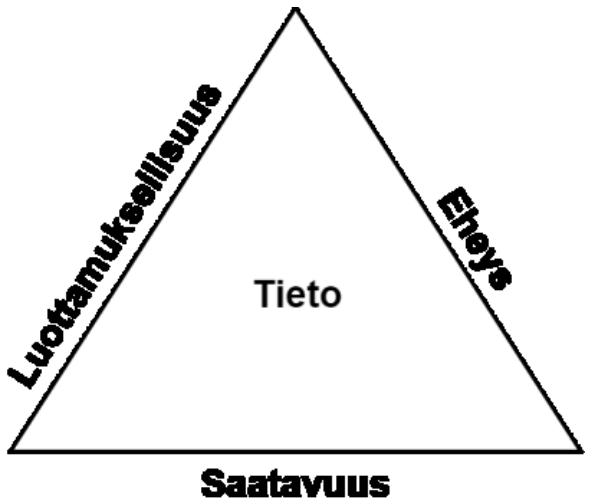
Khokhlov ja Reznik (2018) ovat tehneet samasta aiheesta vastaavan, mutta suppeamman tutkimuksen ja sovelluksen [4]. BlackBerry on myös tehnyt vastaavan kaupallisen ”DTEK by BlackBerry”-nimisen sovelluksen, joka toimii tosin vain BlackBerryn Android-laitteilla [5].

2. TIETOTURVA ANDROIDISSA

Tässä luvussa tutustutaan tietoturvaan yleisellä tasolla ja perehdytään sitten siihen, mitä tietoturva Androidissa tarkoittaa ja miten ja mistä asioista se rakentuu. Lopuksi verrataan myös Androidin tietoturvaa sen suurimpaan kilpailijaan, iOS-käyttöjärjestelmään.

2.1. Tietoturva yleisesti

ISO eli International Organization for Standardization määrittelee tietoturvan olevan ”tiedon luottamuksellisuuden, eheyden ja saatavuuden säilyttämistä” [6]. Luottamuksellisuudella (engl. *confidentiality*) tarkoitetaan, että tietoa ei paljasteta tai anneta saataville luvattomille (engl. *unauthorized*) osapuolille. Eheys (engl. *integrity*) tarkoittaa tiedon virheettömyyttä ja täydellisyyttä. Saatavuudella (engl. *availability*) tarkoitetaan, että tieto on saatavilla ja käytettävissä valtuutetuilla osapuolilla näiden sitä tarvitessa. Näitä kolmen ominaisuuden yhdistelmää kutsutaan usein englanniksi termillä *CIA triad*, CIA-lyhenteen tullessa sanoista *confidentiality*, *integrity* ja *availability*. Tätä kolmikkoa voi havainnollistaa kuvan 1 kolmiolla, jossa tieto on turvattu CIA-kolmikon yhdistelmällä. Näitä kolmea ominaisuutta voidaan pitää tietoturvan kolmena tärkeimpänä pääperiaatteena.



Kuva 1. Tietoturvan muodostava CIA-kolmikko.

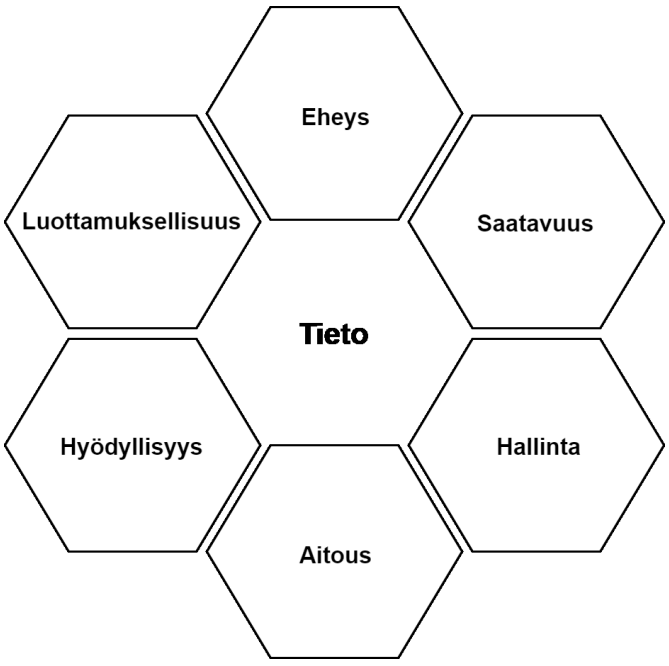
Luottamuksellisuus voi tarkoittaa käytännössä esimerkiksi sitä, että käyttäjä kirjautuu verkkopankkiin katsomaan tilinsä saldon ja tätä tietoa ei vuodeta missään vaiheessa ulkopuolisille tahoille. Jos käyttäjän ja pankin välinen yhteydenpito tässä tapahtumassa onnistutaan kaappaamaan ja purkamaan tai pankin palvelimille tehdään hyökkäys, jossa käyttäjän tiedot onnistutaan varastamaan, luottamuksellisuus on tällöin murrettu.

Eheys tarkemmin määriteltynä tarkoittaa sitä, että estetään tiedon muuttamista luvattomalla ja ei-toivotulla tavalla. Tämänkaltaisiin hyökkäyksiin voidaan lukea luvattoman osapuolen tekemää tiedoston osittaista tai kokonaista poistamista tai

muuttamista. Käyttöjärjestelmätasolla eheyttä voidaan suojella rajoittamalla toimia, joita ohjelmat voivat tehdä tiedostoille. Esimerkiksi Windowsissa ja Linuxissa on käytössä tällaiset järjestelmät [7, s. 5]. Linuxin oman tiedostojen oikeuksien hallintajärjestelmän päälle on kehitelty myös SELinux (Security-Enhanced Linux), joka tiukentaa edelleen näitä oikeuksia [8]. SELinux otettiin käyttöön myös Androidissa sen 4.3-versiosta lähtien [9].

Saatavuudella viitataan siihen, että tietoon päästään tarvittaessa käsiksi, jos siihen on valtuutettu. Saatavuus menetetään, jos polku, jonka kautta tietoon päästään käsiksi, katkeaa tahattomasti tai tahallisesti. Tahattomia saatavuuden estäviä asioita voivat olla esimerkiksi virtakatkot tietojärjestelmässä, sovelluksen tai käyttöjärjestelmän ongelmat tai kaatuminen ja tietoliikenneongelmat. Tahallisia, ulkopuolelta tulevia saatavuuden estäviä hyökkäyksiä kutsutaan yleensä palvelunestohyökkäyksiksi (engl. *denial of service*, *DoS*). [7, s. 6]

CIA-mallia voidaan yksistään käytettynä pitää jossain määrin rajoittuneena ja rajoittavana nykyisessä nopeasti muuttuvassa tietotekniikan maailmassa. Rinnalle onkin kehitetty muita malleja, kuten Donn Parkerin kuusikko (engl. *Parkerian hexad*) [7, s. 7-8]. Siinä CIA-malliin lisätään vielä kolme periaatetta: hallinta, aitous ja hyödyllisyys (engl. *possession, authenticity, utility*). Hallinnalla viitataan tiedon fyysiseen hallitsemiseen ja omistamiseen, aitoudella tiedon alkuperäisyyden todentamiseen eli siihen, että tiedon alkuperäisyys on jäljitettävissä luotettavasti tekijäänsä. Hyödyllisyydellä viitataan siihen, miten tarpeellinen tieto on käyttäjälleen. Tätä mallia on havainnollistettu kuvassa 2.



Kuva 2. Parkerin kuusikko.

Tavallista CIA-mallia voidaan täydentää myös AAA-mallilla (CIA + AAA), jolla saadaan otettua huomioon käyttäjän rooli tietoturvassa. Nämä kolme A-kirjainta tulevat englannin kielen sanoista *authentication*, *authorization* ja *accounting*, jotka voidaan suomeksi kääntää todennukseksi, valtuutukseksi ja kirjanpidoksi. Todennuksella, jossa esimerkiksi annetaan käyttäjätunnus ja salasana, varmistetaan

käyttäjän olevan oikeasti se, joka hän väittää olevansa. Valtuuttamalla annetaan käyttäjälle ne käyttöoikeudet, jotka tälle kuuluvat eikä liian laajoja tai suppeita käyttöoikeuksia. Kirjanpidolla tarkoitetaan sitä, että käyttäjän resurssienkäyttöä valvotaan ja sen tapahtumat kirjataan ylös. AAA-mallia käytetään yleisesti erityisesti verkkopalveluiden pääsynhallinnassa. [10, 11]

Tietoturvaa voi lähestyä myös uhkamallinnuksen kautta. Tähän on kehitetty STRIDE-malli [12], joka pitää sisällään kuusi tietoturvauhkaa: huijaaminen, peukalointi, kiistäminen, tiedon paljastaminen, palvelunesto ja korkeampien käyttöoikeuksien hankkiminen (engl. *spoofing, tampering, repudiation, information disclosure, denial of service, elevation of privileges*). Huijaamisella tarkoitetaan tässä yhteydessä sitä, että hyökkääjä hankkii toisen käyttäjän tiedot käsiinsä ja esiintyy näin toisena käyttäjänä. Peukaloinnilla tarkoitetaan tietojen huomaamatonta muuttamista ja kiistämisellä sitä, että epäluotettava käyttäjä voi tehdä luvattomia toimintoja, kiistää sitten tekemisensä eikä toimintoja voida todistaa tämän käyttäjän tekemiksi. Tiedon paljastumisella tarkoitetaan käyttäjien tietojen paljastumista tahoille, joille niitä ei ole tarkoitettu saataviksi. Palvelunestossa voidaan kaataa tai varata hyökättävä kohde hetkellisesti niin, että varsinaiset käyttäjät eivät pääse käyttämään sitä. Korkeammat käyttöoikeudet hankkimalla hyökkääjä saa käyttöönsä järjestelmän tietoja ja oikeuksia, joihin tällä ei pitäisi olla normaalisti pääsyä. Tällä tarkoitetaan yleensä pääkäyttäjän tai ylläpitäjän (engl. *admin*) oikeuksien oikeudetonta hankintaa.

Löytyneiden uhkien ja hyökkäysmahdollisuuksien arviointiin on olemassa myös valmiita malleja, kuten DREAD [13, s. 93-95] ja Open Web Application Security Project -säätiön (OWASP) riskinmääritysmalli [14]. Esimerkiksi DREAD-mallissa arvioidaan riskin tärkeys sen mahdollisesti aiheuttaman vahingon, toistettavuuden, hyödyntämisen helppouden, mahdollisen vaikutuspiiriin kuuluvien käyttäjien määrän ja löydettävyyden perusteella (engl. *damage, reproducibility, exploitability, affected users, discoverability*).

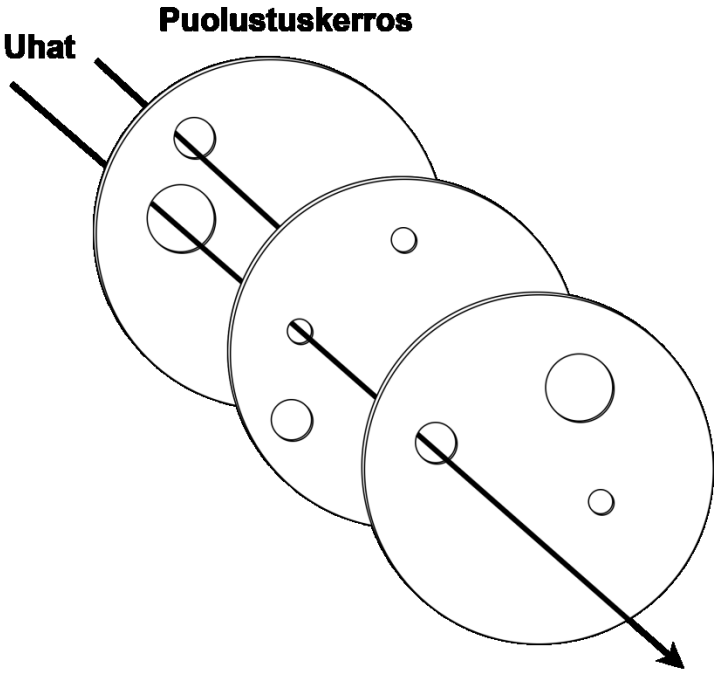
Yhtä yhteistä sääntöä tai mallia tietoturvan ja sen uhkien määrittämiseen ei siis ole. Valmiina löytyy kuitenkin yleisiä, hyväksi havaittuja periaatteita ja malleja. Suurin haaste käytännössä onkin monesti se, mitä mallia kunkin käyttötapauksen tutkimiseen kannattaisi soveltaa.

2.2. Tietoturvan toteutus Androidissa

Androidiin on ajan myötä tullut yhä lisää tietoturvaominaisuuksia ja siihen entisiä ominaisuuksia on korjattu tietoturvan kannalta paremmiksi. Mayrhofer, Stoep, Brubaker ja Kralevich (2019) [15] dokumentoivat ja tarkastelivat julkaisussaan kattavasti Androidin keskeisiä tietoturvaominaisuuksia, kuten prosessien eristämistä toisistaan ja käyttäjän todennusta. Tämä julkaisu toimii hyvänä pohjana Androidin tietoturvan toteutuksen tarkastelussa. Tietoturvamallin Androidissa voidaan katsoa julkaisun perusteella muodostuvan seuraavista asioista: suostumus, todennus, eristäminen, salaus, hyökkäysrajapinnan vähentäminen, järjestelmän eheys ja tietoturvapäivitykset.

Tärkeitä periaatteita Androidin tietoturvan kannalta ovat niin sanotut syvyyspuolustusperiaate (engl. *defense in depth*) ja turvallisen suunnittelun periaate (engl. *safe by design*) [15]. Syvyyspuolustusperiaattella tarkoitetaan sitä, että laitteen suojana on useita kerroksia turvatoimia, jolloin yksittäisellä haavoittuvuudella ei voida murtaa laitteen koko tietoturvaa. Tätä on havainnollistettu kuvassa 3, jossa yksi

tietoturvauhka onnistuu läpäisemään kaikki puolustuskerrokset niissä olevien aukkojen takia. Laitteen ohjelmiston eri komponenttien tulee olla myös suunniteltuja niin, että ne ovat lähtökohtaisesti tietoturvallisia ja suojaavat käyttäjänsä yksityisyyttä, vaikka se rajoittaisi käyttäjää jollain tavalla. Uusia rajapintoja suunnitellessa niiden tulee olla siis tarpeeksi tarkkaan rajattuja, jotta niitä ei voida käyttää myöhemmin väärin.



Kuva 3. Syvyyspuolustuperiaate.

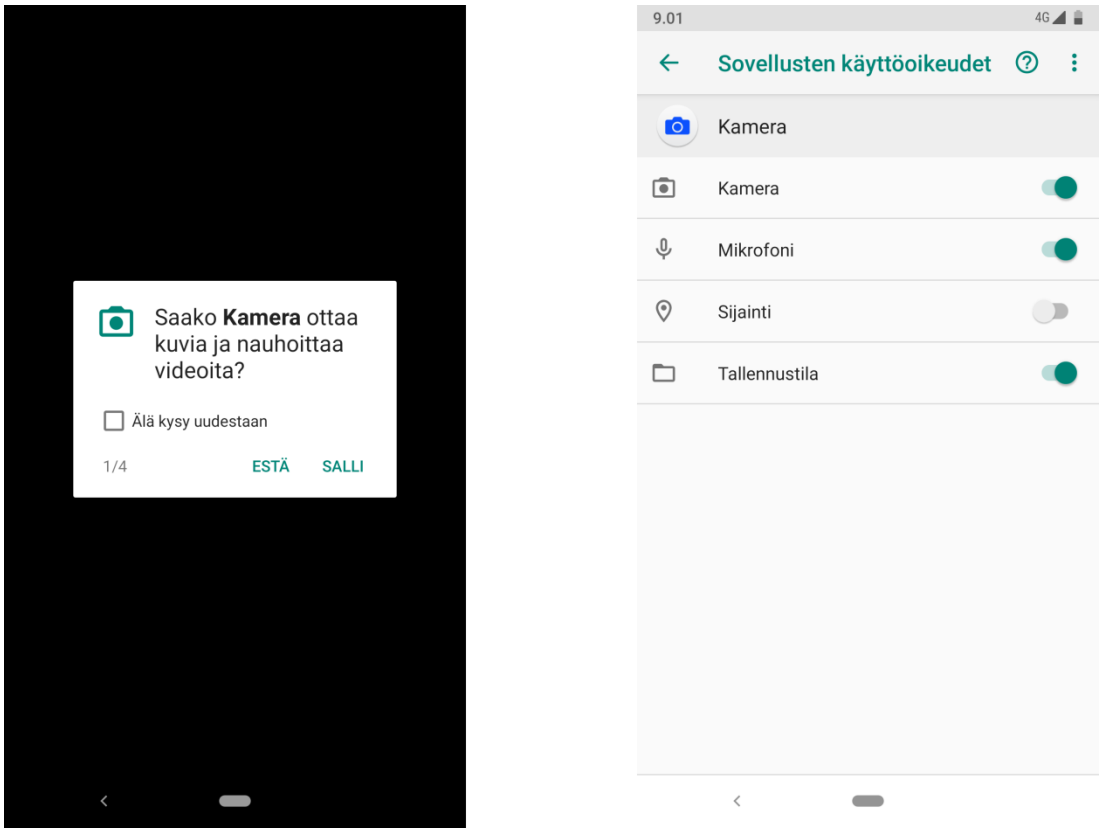
2.2.1. Suostumus

Suostumus (engl. *consent*) Androidin tietoturvan kannalta tarkoittaa sitä, että mitään toimintoa ei suoriteta ilman kolmen päätahon, eli käyttäjän, käyttöjärjestelmän ja kehittäjän (engl. *developer*) suostumusta. Kehittäjällä viitataan tässä laajemmin kaikkiin kehittäjiin, kuten sovelluskehittäjiin ja palveluntarjoajiin. Kuka tahansa näistä tahoista voi käyttää veto-oikeutta estäen toiminnon. [15]

Sovelluskehittäjä voi esimerkiksi määrittää, mihin sovelluksen tietoihin ja rajapintoihin muilla sovelluksilla on pääsy. Sovelluskehittäjä voi antaa käyttäjälle mahdollisuuden jakaa sovelluksen tietoja tarjoamalla kyseisen tiedon jakamiseen soveltuvan rajapinnan [15]. Sovelluksen toimintaa ei myöskään voi muuttaa tältä eikä muilta osin ilman sovelluskehittäjän suostumusta johtuen sovellusten allekirjoittamisesta (engl. *signing*). Jos allekirjoitettua sovellusta muokataan, tämä rikkoo allekirjoituksen, mistä käy ilmi, että sovellus ei ole enää alkuperäisen kehittäjän tekemä [16]. Mikään ei siis kuitenkaan estä käyttäjiä tai kolmansia, mahdollisesti pahanilkisiä osapuolia muokkaamasta sovellusta ja allekirjoittamasta sitä sen jälkeen omilla avaimillaan. Tämän jälkeen sovellusta ei voi kuitenkaan päivittää alkuperäisen sovelluskehittäjän toimittamilla päivityksillä, sillä allekirjoitukset päivityksen ja toisen tahon allekirjoittaman sovelluksen välillä eroavat, eikä sovellusta voi näin ollen päivittää.

Käyttöjärjestelmän suostumus näkyy sen koodin allekirjoituksena, eli järjestelmäkuva (engl. *system image*) on suojattu valmistajan allekirjoituksella. Tätä allekirjoituksen eheyttä suojataan ja tarkistetaan Verified Boot -ominaisuudella (suom. *varmennettu käynnistys*). Näin käyttöjärjestelmä voi varmistaa, että se toimii niin kuin on tarkoitettu ja voi ottaa siten kantaa toimintoihin, joita laitteella tehdään. [15, 17]

Käyttäjän suostumusta kyselevät esimerkiksi sovellukset käyttöoikeuksien (engl. *permission*) muodossa silloin kun ne tarvitsevat pääsyä laitteen tiedostoihin tai laitteistokomponentteihin, kuten kameraan tai mikrofoniin. Androidin kehityksessä on parannettu jatkuvasti käyttöoikeuskyselyiden ymmärrettävyyttä ja niiden määrää on myös pyritty rajoittamaan, ettei käyttäjä väsy ja sokeudu jatkuville kyselyille [15]. Toisaalta yksinkertaistamalla suostumusmallia on jouduttu rajoittamaan käyttäjän mahdollisuuksia tehdä syvemmälle meneviä muutoksia [15]. Käyttäjän annettua tai evättyä suostumuksensa johonkin hänellä on myös mahdollisuus aina perua tehty päätös myöhemmin sovellusten asetuksista. Käyttöoikeuskyselyä ja niiden hallinnointia on esitelty kuvassa 4.



Kuva 4. Käyttöoikeuksien kysyminen vasemmalla, oikealla niiden hallinnointinäkymä.

2.2.2. Todentaminen

Todentamisella eli autentikoinnilla (engl. *authentication*) varmistetaan, että laitetta käyttää sen omistaja tai omistajan valtuuttama henkilö. Tämä todentaminen toteutetaan yleisimmin näytön lukituksella (engl. *lockscreen*). Sen lisäksi voi olla

käytössä muita todentamismenetelmiä, kuten sormenjälki, kasvojentunnistus ja tunnettuun Bluetooth-laitteeseen yhteydessä oleminen. Nämä tosin lasketaan heikommiksi todentamismenetelmiksi kuin varsinainen näytön lukitus esimerkiksi PIN-koodilla tai salasanalla. Android-laite voi pyytää välillä varsinaisen näytön lukitustavan käyttämistä, jos puhelin on avattu viime kerroilla vain heikompaa todentamismenetelmää käyttäen. [15, 18]

Näytön lukitus hidastaa tietenkin laitteen käyttöä verrattuna siihen, että sitä voisi käyttää ilman mitään suojausta. Tällöin laitteen kaikki toiminnot olisivat tosin kenen tahansa käytössä, joka pääsee laitteeseen fyysisesti käsiksi. Siksi näytön lukitusta voidaan pitää välttämättömänä pahana laitteen suojaamiseksi. Laitteen lukituksen avaamista helpottavat kuitenkin nopeammat tavat todentaa käyttäjä, kuten edellä mainitut sormenjälki- ja kasvojentunnistus.

2.2.3. Eristäminen ja hallinta

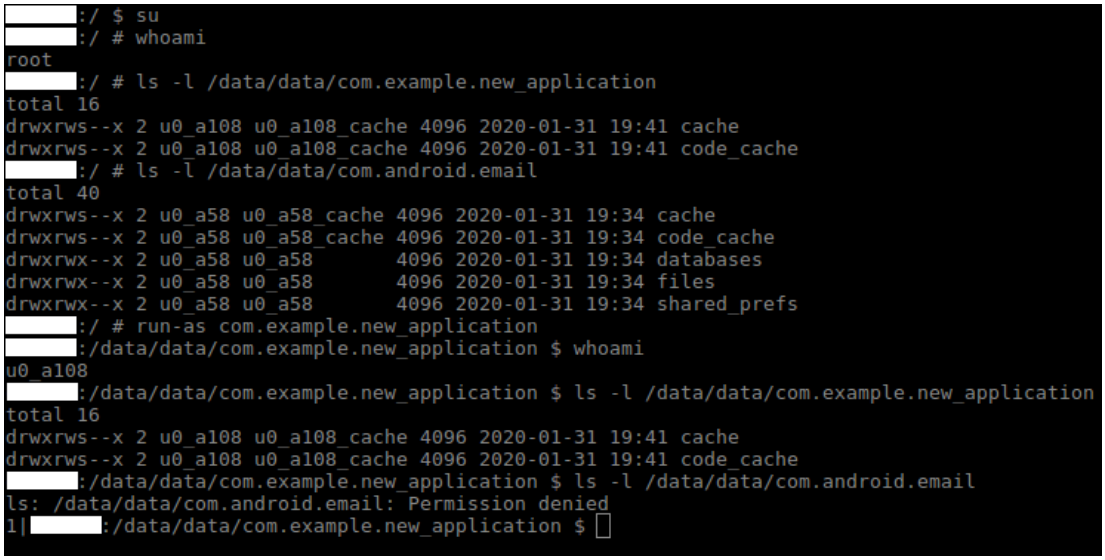
Suojautumista haitalliselta koodilta ajon aikana (engl. *runtime*) toteutetaan Androidissa hiekkalaatikoinnin ja pääsynvalvonnan avulla (engl. *access control*). Androidissa pääsynvalvonta on toteutettu seuraavilla mekanismeilla: yksilöpohjainen pääsynvalvonta (engl. *Discretionary Access Control, DAC*), sääntöpohjainen pääsynvalvonta (engl. *Mandatory Access Control*, *MAC*) ja Androidin käyttöoikeudet (engl. *Android permissions*). Hiekkalaatikointi voidaan jakaa sovellusten, järjestelmäprosessien, kernelin ja kernelin alla olevien komponenttien hiekkalaatikointiin. [15]

Yksilöpohjaisessa pääsynvalvonnassa objektin omistaja (käyttäjä tai prosessi) määrittää itse kuka pääsee käsiksi niihin ja millä tavalla (esimerkiksi luku-, kirjoitus- ja suoritusoikeudet). Sääntöpohjaisessa pääsynvalvonnassa objektin omistaja ei voi itse päättää kenellä siihen on pääsy. Siinä pääsyoikeudet määrää ryhmä tai yksilö, jolla on määräysvalta niiden määrittämiseen. Tästä esimerkkinä voivat olla valtion organisaatiot, joissa pääsy tiettyihin dokumentteihin voi olla rajattu dokumentin suojaustason, dokumenttia pyytävän käyttäjän turvallisuusluokituksen ja käyttäjän todellisen tarpeen perusteella. Jos käyttäjällä ei esimerkiksi ole tehtäviensä takia tarvetta saada käyttää dokumenttia, pääsy siihen voidaan evätä. [7, s. 42-43]

Androidin käyttöoikeudet suojaavat arkaluonteisia tietoja ja palveluita sovelluksilta, eli sovellukset eivät pääse niihin suoraan käsiksi ilman erillistä suostumusta tai oikeutusta [15]. Käyttöoikeudet määritellään sovelluksen manifestitiedostossa, joka tarkoittaa jokaisen Android-sovelluksen (engl. *application*) mukana olevaa ”AndroidManifest”-nimistä XML-tiedostoa (Extensible Markup Language). Se kertoo olennaiset tiedot sovelluksesta, muun muassa sovelluksen nimen, sen komponentit, kuten aktiviteetit, palvelut ja vastaanottimet sekä sovelluksen tarvitsemat käyttöoikeudet [19].

Hiekkalaatikointi sovellusten osalta Androidissa perustuu suurimmaksi osaksi yksilöpohjaiseen pääsynvalvontaan (DAC). Sillä annetaan joka sovellukselle oma Unix-käyttäjätunniste eli UID (engl. *user identifier*, *UID*) ja kansio tiedostoja varten, johon muilla sovelluksilla ei ole pääsyä. Tätä on esitelty kuvassa 5, jonka alussa katsotaan pääkäyttäjän (käyttäjätunniste root) oikeuksilla sähköpostisovelluksen (käyttäjätunniste u0_a58) kansion sisältöä, mutta johon toisella sovelluksella (käyttäjätunniste u0_a108) ei ole oikeuksia. Yksilöpohjaisen pääsynvalvonnan mallissa järjestelmätason prosessit ovat tosin hiekkalaatikon ulkopuolella. Sen puutteita on korjattu Androidin kehittyessä monilta osin, esimerkiksi ottamalla

käyttöön sääntöpohjainen pääsynvalvonta, ajonaikaiset pääsyoikeudet (engl. *runtime permissions*) ja pienentämällä hyökkäyspinta-alaa (engl. *attack surface reduction*). [15]



Kuva 5. Yksilöpohjainen pääsynvalvonta toiminnassa.

Järjestelmäprosessit on myös eristetty toisistaan hiekkalaatikoimalla ne saman tapaan kuin tavalliset sovellukset. Tämä oli tarpeen esimerkiksi median kehystoimintojen, wifin ja Bluetoothin puolelta tulevien uhkien torjumiseen [15]. Järjestelmäprosessien eristykseen on otettu käyttöön myös sääntöpohjainen pääsynvalvonta SELinuxin avulla [9].

Kernelin eli ytimen tarjoamat rajapinnat ovat nostaneet houkuttelevuuttaan hyökkääjien silmissä samalla kun käyttäjäavaruutta (engl. *userspace*) on kovennettu tietoturvan kannalta. Suurin osa kernelin haavoittuvuuksista johtuu laitteistoajureista, joten tätä tilannetta on korjattu vähentämällä prosessien pääsyä ajureihin ja hiekkalaatikoimalla itse koodia kernelissä. [15]

Jos kernel ja kaikki sen yläpuolella olevat kerrokset onnistutaan murtamaan hyökkäyksessä, kernelin alapuolella on myös vielä suojauskeinoja tärkeimpien komponenttien ja salausavainten turvana. Näihin lukeutuvat mahdollisuus käyttää laitteistopohjaista avainvarastoa (engl. *hardware-backed keystore*) [20], mahdollinen erillinen peukaloinnilta suojattu piiri (Secure Element) kryptograafista dataa varten [21] ja lukitusnäytön avaamisen todentavat Gatekeeper [22] ja luotettu ohjelmistokoodien ajoympäristö (engl. *trusted execution environment, TEE*). [15]

2.2.4. Salaus

Salaus voidaan jakaa levossa olevan tiedon ja liikkeessä olevan tiedon salaamiseksi. Levossa olevalla tiedolla tarkoitetaan tässä tietoa, joka on laitteen haihtumattomassa muistissa (engl. *non-volatile memory*). Mahdollisia tapoja päästä käsiksi siihen ovat lukea muistia silloin kun kernel ei ole ajossa, esimerkiksi kun laite on sammutettuna, tai ohittaa kernel jotenkin ja päästä lukemaan muistia suoraan. Liikkeellä olevalla tiedolla tarkoitetaan dataa, joka liikkuu internetissä ja paikallisissa verkoissa kun laitteet kommunikoivat toistensa kanssa. [15]

Levossa oleva tieto on Android 5.0 -käyttöjärjestelmäversiosta lähtien mahdollista suojata käyttäen koko levyn salausta (engl. *full disk encryption, FDE*) [23]. Sen puutteita korjattiin mahdollistamalla tiedostopohjainen salaus (engl. *file-based encryption, FBE*) Android 7.0 -käyttöjärjestelmäversiosta eteenpäin [24] ja tämän lisäksi tuotiin Android 9:stä lähtien tuki tiedostojärjestelmän metadatan salaamiselle (engl. *metadata encryption*) [25]. Metadataa ovat esimerkiksi kansioiden nimet, tiedostojen koot, niiden käyttöoikeudet ja luontiajat. Näiden salaustapojen toimintatavoista on kerrottu tarkemmin kappaleessa 3.10. Salauksen ollessa käytössä muistissa olevat tiedot on helppo tehdä lukukelvottomiksi ja käytännössä mahdottomiksi palauttaa poistamalla salaukseen käytetty avain [15]. Näin koko tallennustilaa ei tarvitse tyhjentää.

Androidin kehityksessä oletetaan, että kaikki verkot, johon laite yhdistetään, ovat vihamielisiä tai niiden liikennettä vakoillaan [15]. Näin ollen lähtökohtaisesti kaikki liikkeellä oleva tieto salataan ”päästä päähän” (engl. *end-to-end encryption*). Verkkoliikenteessä kulkevaa tietoa voidaan suojata TLS-protokollalla [26], ja salaamaton HTTP-liikenne on oletuksena estetty sovelluksilta [27].

2.2.5. Haavoittuvuuksien ehkäisy

Suurin osa Androidin haavoittuvuuksista on johtunut suojaamattomasta pääsystä muistiin [28]. Tätä kautta tulevia uhkia on ehkäisty Androidissa muistin osoiteavaruuden sijoittelun satunnaistamisella (engl. *address space layout randomization, ASLR*), muistin luku-, kirjoitus- ja suoritusoikeuksien rajoituksilla sekä puskurien ylivuotojen suojaamisella [15].

Haavoittuvuuksia ehkäistään Androidissa myös muilta osa-alueilta, pääpainona ne alueet, jotka ovat saatavilla etänä, kuten median kehystoiminnot ja alueet, joiden kautta on aiemmin paljastunut paljon haavoittuvuuksia [15]. Näitä haavoittuvuuksia etsitään ja ehkäistään tarjoamalla haavoittuvuuksien metsästysohjelmia (engl. *bug bounty program*) ja käyttämällä erilaisia työkaluja virheiden aikaiseen havaitsemiseen, kuten UndefinedBehaviorSanitizer [29]. Androidissa käytetään myös LLVM-kompiloijan Control Flow Integrity (CFI) -suojausta, jolla estetään muutokset käännettyjen binääritiedostojen alkuperäiseen kontrollivuohon, joka mahdollistaisi erilaisia hyökkäyksiä [15, 30].

Haavoittuvuuksia ehkäistään ja etsitään siis aktiivisesti. Nämä ehkäisytavat yhdessä aiemmin mainittujen eristämismekanismien kanssa muodostavat puolustukseen kerroksia. Jos yksi näistä kerroksista pettäisi, muut voivat vielä estää ja rajoittaa mahdollisen haavoittuvuuden aiheuttamaa uhkaa.

2.2.6. Järjestelmän eheys

Android-laitteen järjestelmän eheys on olennainen kerros syvyyspuolustautumisessa (engl. *defense in depth*). Järjestelmän eheyttä ja peukaloimattomuutta tarkastellaan Androidissa Verified Boot -ominaisuudella, joka puolestaan käyttää Linux-kernelin dm-verity -ominaisuutta (device-mapper-verity) [15]. Sillä tarkistetaan luottamusketju (engl. *chain of trust*) lähtien laitteistosuojatusta luottamuksen ytimestä (engl. *root of trust*) käynnistyslataajaan (engl. *bootloader*), käynnistysosioon ja muihin todennettuihin osioihin, kuten system- ja vendor-osiot [17]. Jos niissä on pieniä virheitä, Android voi korjata niitä itsekseen tiettyyn rajaan

asti virheenkorjauskoodin (engl. *forward error correction*) ansiosta [31]. Verified Boot -ominaisuus tarkistaa myös Androidin version tilan niin, ettei laitteelle ole asennettu vanhempaa, oikealla tavalla allekirjoitettua käyttöjärjestelmää. Tätä ominaisuutta Verified Boot -ominaisuuden sisällä kutsutaan nimellä *rollback protection* [32]. Vanhemmassa käyttöjärjestelmässä voi olla haavoittuvuuksia, joita voidaan käyttää hyväksi erilaisissa hyökkäyksissä. Verified Boot -ominaisuuden tila on myös sovellusten tarkistettavissa avainten vahventamisen kautta (engl. *key attestation*) [33].

Muiden kuin järjestelmätason sovellusten eheyttä suojellaan niiden allekirjoittamisella. Sovelluksia ei voi muuttaa eikä päivittää, jos uuden, muokatun sovelluksen allekirjoitus ei täsmää edellisen kanssa. [34]

2.2.7. Tietoturvapäivitykset

Säännöllinen haavoittuvuuksien paikkaaminen päivityksillä voidaan nähdä yhtenä tietoturvan puolustuksen kerroksista [15]. Haasteena Androidin päivityksissä on kuitenkin laajalle sen jakaantunut ekosysteemi: laitevalmistajia on useita ja erilaisia laitteita valtava määrä. Päivitysten tekemistä Android-laitteille on helpotettu Treble-projektilla, joka julkaistiin Android 8 -version yhteydessä [35]. Sen myötä valmistajien on helpompi päivittää laitteensa uusimpiin Android-versioihin ilman raskaita muutoksia omiin ohjelmistoihinsa.

2.3. Androidin tietoturva verrattuna muihin käyttöjärjestelmiin

Tietokonekäyttöjärjestelmien ja mobiilikäyttöjärjestelmien tietoturvan toteutusten vertaaminen on haasteellista, sillä niiden tietokoneiden ja mobiililaitteiden käyttötarkoitukset eroavat monilta osin toisistaan. Mobiililaitteet ovat yleensä ominaisuuksiltaan monipuolisempia (matkapuhelinverkot, NFC, kamerat, GPS, erilainen sovellusvalikoima) ja niitä kuljetetaan nimensä mukaisesti enemmän mukana kun taas tietokoneet ovat yleensä enemmän paikallaan, kannettavat toki vähemmän kuin pöytäkoneet. Mobiililaitteissa on monipuolisempien ominaisuuksien myötä enemmän pinta-alaa ja rajapintoja, joita suojata. Tietokoneet voivat yleensä käyttää myös kiinteitä Ethernet-yhteyksiä, jotka voidaan nähdä lähtökohtaisesti turvallisempina kuin mobiililaitteiden langattomat yhteydet.

Tietokoneiden ja mobiililaitteiden tietoturvasta löytyy silti paljon myös yhtenäisyyksiä, muun muassa mahdollisuus käyttää lukitusnäyttöä ja tallennustilan salausta, mutta mobiililaitteiden ja tietokoneiden vertaaminen on silti hieman sama asia kuin henkilöautojen ja kuorma-autojen vertailu. Ne on tehty eri käyttötarkoituksiin.

Androidin tärkein haastaja mobiilikäyttöjärjestelmien suosiossa on iOS, kuten johdannon alussa todettiin. Se on suljetun lähdekoodin käyttöjärjestelmä toisin kuin Android. Android-laitteiden ohjelmistot perustuvat avoimen lähdekoodin Android Open Source Project -projektista (AOSP) rakennetuille käyttöjärjestelmille, mutta niissä on tämän lisäksi lähes poikkeuksetta myös suljettua lähdekoodia, kuten laitevalmistajan omia ohjelmia sekä ajureita. Android-laitteiden osittaisen avoimuuden takia niistä on helpompi löytää haavoittuvuuksia AOSP-projektin koodia tutkimalla. Tämä on sekä hyvä että huono puoli avoimessa lähdekoodissa,

sillä sen koodia voi vapaasti tutkia kuka vain ja kuka vain voi myös ehdottaa korjauksia löytämilleen virheille ja haavoittuvuuksille.

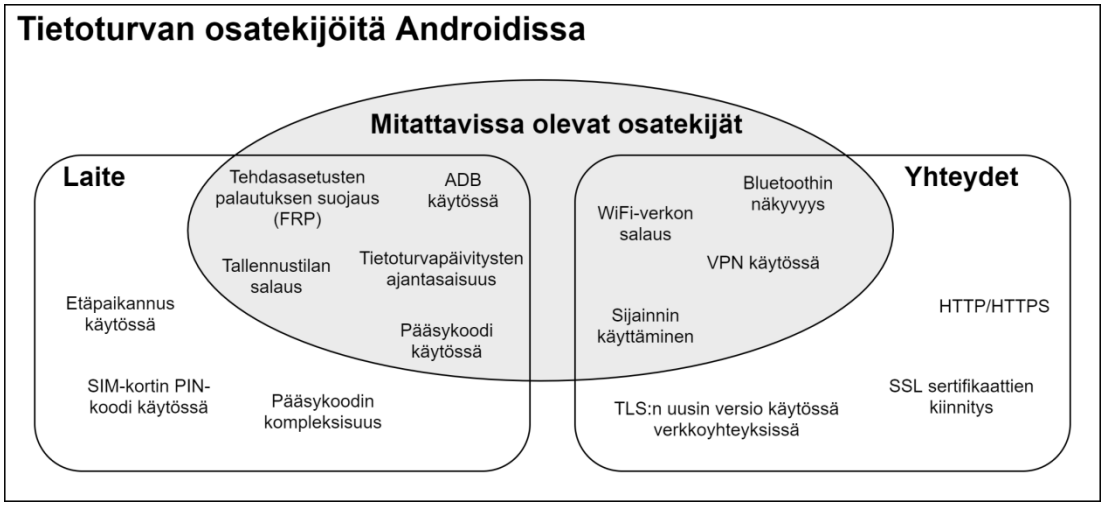
Suljetussa lähdekoodissa samanlaista ongelmaa ei ole, mutta siitä voi löytyä yhtä hyvin haavoittuvuuksia. Niiden löytäminen on vain paljon vaikeampaa muille kuin suljetun lähdekoodin omistajalle, mutta samalla koodi ei ole todennäköisesti yhtä koeteltua ja katselmoitua kuin avoin lähdekoodi.

Sekä Androidissa että iOS:ssa on paljon yhteisiä tietoturvaominaisuuksia, kuten tallennustilan salaus, sovellusten allekirjoittaminen, niiden hiekkalaatikointi ja tarkistaminen sovelluskauppaan ladattaessa. Mohamedin ja Patelin (2015) julkaiseman Androidin ja iOS:n tietoturvaa vertailevan tutkimuksen [36] mukaan tärkeimmät erot käyttöjärjestelmien välillä iOS:n eduksi ovat Androidin sovellusten mahdollisuus allekirjoittaa sovelluksia uudelleen ja tarkempi sovellusten käyttöoikeuksien luottaminen käyttäjille. Sovelluskehittäjien tekemät alkuperäiset sovellukset on mahdollista purkaa ja muuttaa niiden toimintaa, jonka jälkeen ne voidaan allekirjoittaa omalla avaimella. Sovelluksen alkuperän varmistaminen jää käyttäjän ja mahdollisen sovelluskaupan tehtäväksi. iOS-laitteille sovelluksia ei voi asentaa kuin Applen oman App Store -sovelluskaupan kautta toisin kuin Androidissa, jossa käyttäjälle annetaan vapaat kädet tämän suhteen, tosin varoitusten kanssa. Sovellusten käyttöoikeudet ovat Androidissa myös tarkemmin jaoteltuja ja niiden myöntäminen luotetaan käyttäjälle, kun taas iOS:ssa niitä on yksinkertaistettu ja kavennettu jo oletusarvoisesti.

Android-käyttäjät saavat vapaammat kädet säätää laitteidensa tietoturvaa kun taas iOS-laitteiden käyttäjillä ei ole samanlaista valinnanvaraa mutta lähtökohtaisesti iOS:n voidaan katsoa olevan turvallisempi tavalliselle käyttäjälle. Android-käyttäjän tulee olla valveutuneempi ja esimerkiksi perehtyä laitetta hankkiessa laitevalmistajien päivitysten julkaisun jatkuvuuteen ja olla tietoisempia asentamiensa sovelluksien alkuperästä ja luotettavuudesta kuin iOS-laitteiden käyttäjien.

3. ANDROIDIN TIETOTURVAN MITATTAVIA OSATEKIJÖITÄ

Android-laitteen tietoturvaan vaikuttavat monet tekijät, joihin käyttäjä voi vaikuttaa omilla toimillaan. Tällaisia ovat esimerkiksi näytön lukituksen ja sijaintipalveluiden käyttäminen. Näitä tietoturvaan vaikuttavia osatekijöitä voi eritellä Android-laitteista todella paljon, mutta kaikkia niistä ei voi mitata Androidin sovellusten ohjelmointirajapintaa käyttämällä. Tätä ongelmaa havainnollistetaan kuvassa 6, jossa muutamia esimerkkejä tietoturvan osatekijöistä on jaoteltu karkeasti laite- ja yhteyskategorioiden alle. Näiden osatekijöiden tilasta vain osan voi mitata sovellusten ohjelmointirajapinnan kautta. Valmista listaa näistä tietoturvan osatekijöistä ei ole olemassa.



Kuva 6. Mitattavien tietoturvan osatekijöiden suhde kaikkiin osatekijöihin.

Tässä työssä keskitytäänkin niihin tietoturvan osatekijöihin, jotka voidaan järkevällä tavalla mitata laitteesta, joihin käyttäjä voi itse vaikuttaa ja joihin Androidin sovellusten ohjelmointirajapinta antaa pääsyn. Tarkemmin näitä osatekijöitä esitellään tässä luvussa 11 kappaletta ja näiden 11 osatekijän tarkistamiseen tarvittavat mittausmetodit toteutettiin tässä työssä tehtyyn sovellukseen.

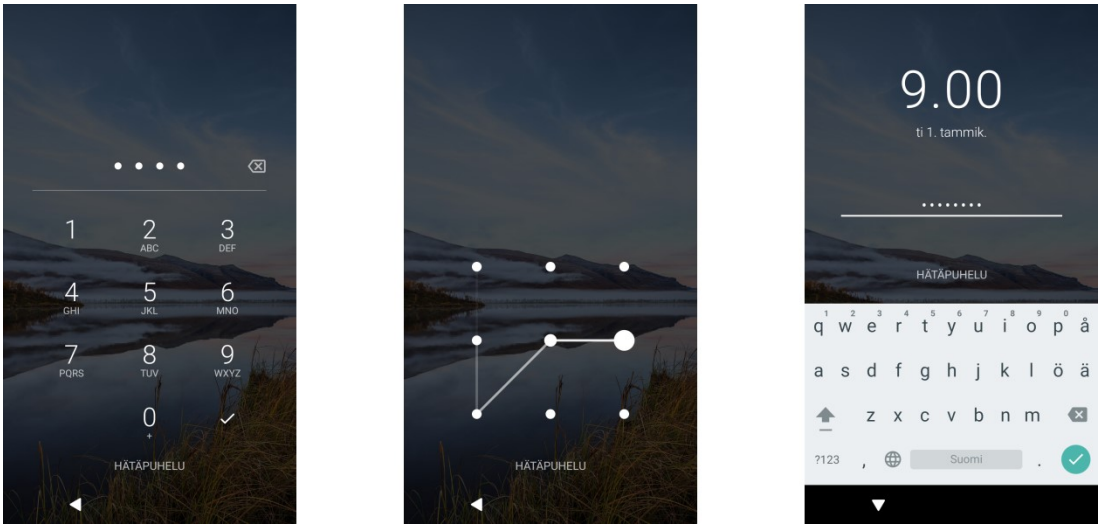
Jokainen osatekijä selitetään auki, minkä jälkeen kerrotaan siitä koituvat hyödyt (jos ne eivät ole ilmeisiä), haitat ja mahdolliset hyökkäystavat. Lopuksi esitellään miten osatekijä otetaan käyttöön, jos se on tietoturvaominaisuus, tai miten sen luomaa uhkaa ehkäistään (engl. *threat mitigation*), jos se on tietoturvauhka. Tässä työssä ei tutkita sellaisia tietoturvan osa-alueita, joiden tietoturvauhan määrittäminen vaatisi tiedon keräämistä ja analysointia (esimerkiksi haitallisten sovellusten tunnistaminen tietokantaan vertaamalla). Myös yksittäisten CVE-raportoitujen (Common Vulnerabilities and Exposures) haavoittuvuuksien tarkastelu on rajattu tarkastelun ulkopuolelle.

Tässä luvussa tehty osatekijöiden selittäminen ja perustelu hyödyttää myös tehdyn sovelluksen kehittämistä, sillä osatekijöiden mittaamisen perusteet, tilat ja mahdolliset korjaustoimenpiteet tulee myös olla selitettävissä käyttäjille. Pelkkä osatekijöiden ja niiden tilojen listaaminen ei välttämättä riitä kuin edistyneemmille Android-käyttäjille, jotka ovat myös perehtyneet niiden tietoturvaominaisuuksiin.

Samoin käyttäjän vaikutettavissa olevien osatekijöiden korjaaminen voi jäädä tekemättä, jos käyttäjällä ei ole mitään ohjetta miten se tehdään.

3.1. Näytön lukituksen käyttäminen

Android-laitteille voidaan kytkeä näytön lukitus joko PIN-koodilla, kuviolla tai salasanalla. PIN-koodi on vähintään nelinumeroinen numerosarja. Kuvioon kuuluu 3x3 pistettä, joista vähintään neljä pistettä yhdistetään piirtämällä sormella. Salasana taas tarkoittaa vähintään neljää alfanumeerista merkkiä tai erikoissymbolia. Nämä vaihtoehdot näkyvät kuvassa 7. Vasemmalla kuvassa on esitetty PIN-koodi, keskellä kuvio ja oikealla salasana. Näytön lukituksen avaaminen on mahdollista myös sormenjäljellä tai kasvojentunnistuksella, jos laite tukee niitä. Tällöin laitteelle pitää olla kuitenkin lisäksi määriteltynä PIN-koodi, kuvio tai salasana siltä varalta, että laitteen sormenjälkitunnistin tai kasvojentunnistukseen käyttämä kamera vioittuu. Nämä näytönlukitusvaihtoehdot esitetään Googlen Android-tukisivuilla [37].



Kuva 7. Näytön lukituksen vaihtoehdot.

Näytön lukituksen voi myös jättää asettamatta, mutta tällöin kaikkia laitteen suojausominaisuuksia, kuten tallennustilan salausta, ei voi käyttää. Lukituksen asettamatta jättäminen antaa kenelle tahansa pääsyn laitteen asetuksiin, sovelluksiin ja tiedostoihin, kuten kuviin ja tekstidokumentteihin. Joihinkin sovelluksiin, esimerkiksi kuvagalleriasovelluksiin, voi asettaa erillisen oman salasanan, jotta niitä pääsee käyttämään. Tämän pystyy kuitenkin kiertämään käyttämällä toista sovellusta tai asentamalla sellaisen, joka antaa pääsyn haluttuihin tiedostoihin ilman erillistä salasanaa. Samalla tavalla puhelimeen voi asentaa minkä tahansa sovelluksen esimerkiksi käyttäjän vakoilua varten. Laitteen tiedostojen ja tietoturvan suojaamiseksi näytön lukitus voidaan katsoa ensiarvoisen tärkeäksi. Mitä monimutkaisempi PIN-koodi, kuvio tai salasana on käytössä, sitä paremmin käyttäjän tiedot ovat turvassa.

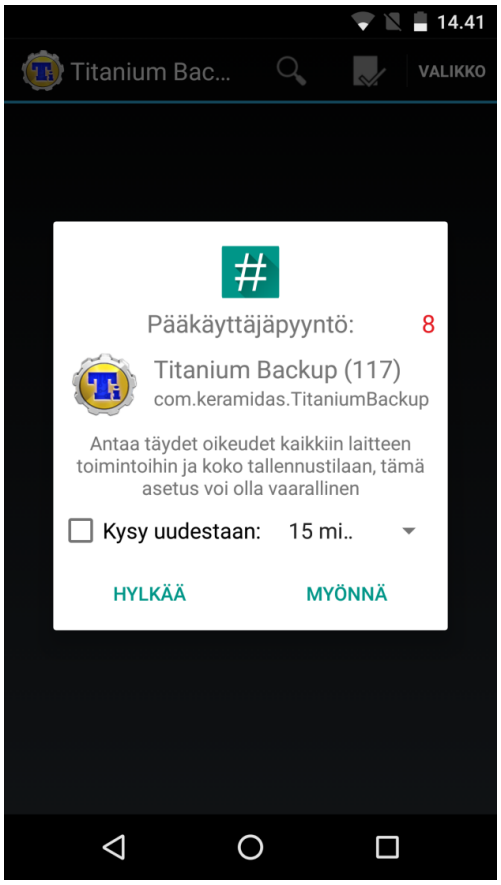
3.2. Laitteen roottaaminen

Roottaus (engl. *rooting*) tarkoittaa Android-laitteissa superuser- eli pääkäyttäjätason oikeuksien hankkimista, jolloin käyttäjä saa oikeudet kaikkien tiedostojen muokkaamiseen ja pääsyn kaikkiin käyttöjärjestelmän komentoihin. Superuser-käyttäjää käytetään Unix-tyylisissä käyttöjärjestelmissä [38] ja Androidin voidaan myös katsoa olevan Unix-pohjainen. Superuser-käyttäjää kutsutaan usein myös root-käyttäjäksi [38] ja tästä tulee nimitys roottaminen.

Root-tason oikeuksilla käyttäjä voi muokata käyttöjärjestelmää sen kaikilla tasoilla. Tästä on monenlaista hyötyä käyttäjälle. Rootatussa Android-laitteessa voi muokata normaalikäyttäjälle saavuttamattomissa olevia järjestelmätiedostoja ja näin muuttaa laitteen toimintaa ja esimerkiksi käyttöjärjestelmän ulkonäköä omalla teemalla ja käynnistysanimaatiolla. Roottauksen avulla voi myös poistaa laitevalmistajan esiasentamia sovelluksia, joista ei ole käyttäjälle välttämättä hyötyä (engl. *bloatware*) mutta joita ei voi poistaa normaalikäyttäjänä. Rootattu käyttäjä voi myös tehdä laitteesta täydellisen varmuuskopion ja palauttaa varmuuskopioita, asentaa omia kerneleitä, käyttöjärjestelmäversioita sekä sovelluksia, jotka vaativat root-oikeuksia. Tällaisia sovelluksia ovat esimerkiksi koko laitteen kattavat mainostenesto-ohjelmat sekä yli- ja alikellotusohjelmat. Ylikellottamalla voidaan parantaa laitteen suorituskykyä ja alikellottamalla voidaan pidentää sen akunkestoa. [39, s. 57]

Haittapuolia roottauksesta syntyy tietoturvan ja mahdollisesti myös toimintavarmuuden kannalta. Laitetta rootatessa sen ohjelmisto on mahdollista rikkoa niin, että sitä ei voi palauttaa toimivaksi (engl. *bricking*). Myös rootattua laitetta syvemmällä tasolla (esimerkiksi käyttöjärjestelmä, prosessorin kellotaajuus) muokkaavat säädöt voivat saada laitteen toimimaan epästabiilisti. Se voi uudelleenkäynnistyä odottamattomasti tai sovellukset voivat kaatuilla.

Tietoturvaa ajatellen roottaaminen tekee paljon aukkoja Androidin tietoturvamalliin. Se rikkoo sovellusten eristyksen eli niin kutsutun hiekkalaatikoinnin. Rootatussa laitteessa sovellukset voivat pyytää käyttäjältä laajempia oikeuksia, jolloin ne pystyvät muokkaamaan ja lukemaan myös toisten sovellusten tietoja ja muita käyttöjärjestelmän tiedostoja, jotka ovat normaalisti sovellusten tavoittamattomissa [2, 40]. Kuvassa 8 on kuvakaappaus sovelluksen pyynnöstä pääkäyttäjäoikeuksiin eli root-tasolle. Oikeuksia pyytävä sovellus on varmuuskopiointiin ja niiden palauttamiseen tarkoitettu Titanium Backup [41] ja pääkäyttäjäpyyntöjä hallinnoiva sovellus on nimeltään SuperSU [42].



Kuva 8. Sovellus pyytää pääkäyttäjän oikeuksia rootatussa laitteessa.

Rootatussa puhelimessa olevat haittaohjelmat voivat siis saada laajemmat oikeudet kuin roottaamattomissa, jos käyttäjä ei ole tarkkana mitä sovelluksia laitteeseen asentaa ja mille niistä antaa pääkäyttäjätason oikeuksia. Haittaohjelmat pystyvät tekemään näillä oikeuksilla paljon enemmän vahinkoa verrattuna siihen, että ne olisi eristetty toisista sovelluksista ja käyttöjärjestelmän syvemmistä toiminnoista. Rootatut laitteet eivät myöskään välttämättä saa enää laitteen valmistajan toimittamia ohjelmisto- ja tietoturvapäivityksiä. Tällöin niistä tulee ajan kanssa yhä haavoittuvaisempia erilaisia hyökkäyksiä vastaan [43]. Rootattu laite voi silti saada edelleen päivityksiä. Tämä riippuu tavasta, jolla laite on rootattu ja siitä, onko laitteelle asentanut oman käyttöjärjestelmäversion alkuperäisen tilalle. Jos laitteelle on asentanut oman käyttöjärjestelmäversion, sille voi tulla päivityksiä käyttöjärjestelmäversion tekijältä. Esimerkiksi LineageOS, joka on suosittu avoimen lähdekoodin käyttöjärjestelmä Android-laitteille, saa päivityksiä säännöllisesti [44].

Jotkin sovellukset tarkistavat onko laite rootattu ja estävät sovelluksen tai sen joidenkin osien käytön, jos laite todetaan rootatuksi. Näin tekevät esimerkiksi monet verkkopankkisovellukset, sillä ne haluavat suojella käyttäjiensä tietoturvaa. Roottauksen tila on kuitenkin mahdollista piilottaa sitä kysyviltä sovelluksilta esimerkiksi muokkaamalla Androidin API-rajapintoja (Application Programming Interface), joita sovellukset käyttävät tarkistuskyselyn tekemiseen. [45]

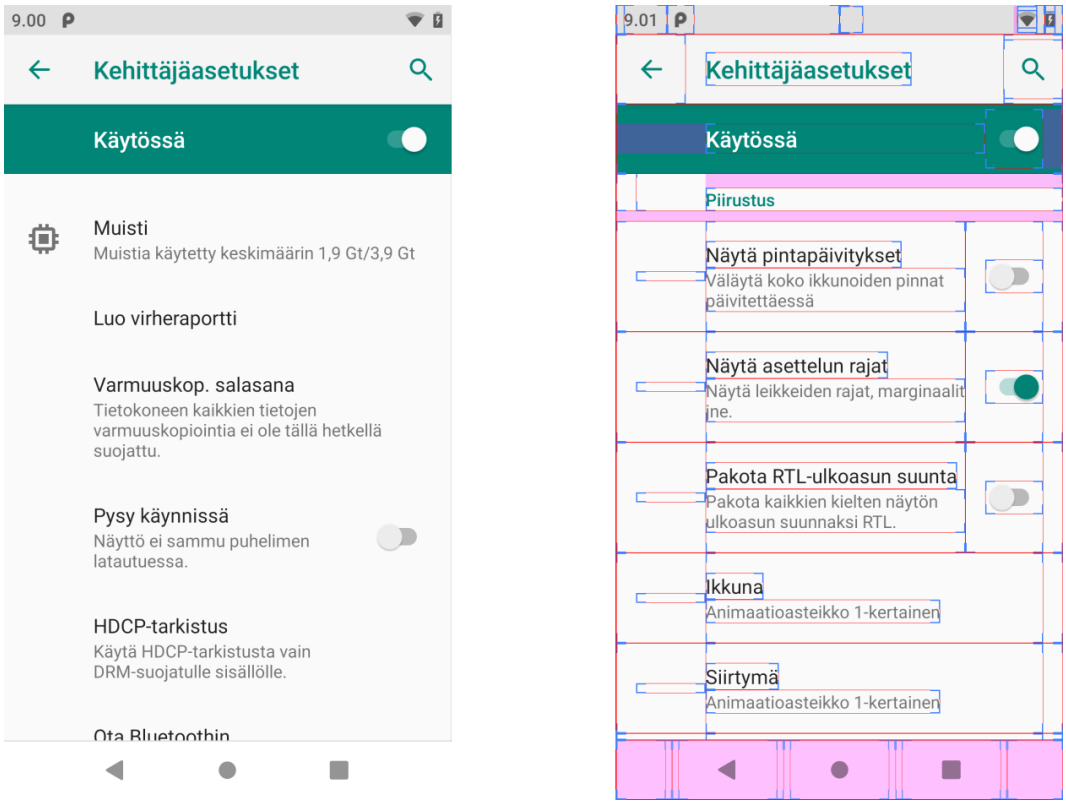
Yksi yleinen uskomus roottamisen vaikutuksista Android-laitteen takuuseen on, että roottaminen lopettaa automaattisesti takuun vaikka takuuaikaa olisi vielä jäljellä. Tämä ei kuitenkaan pidä paikkaansa ainakaan EU-maissa. EU-tasolla on säädetty direktiivi, jonka mukaan laitteen valmistaja on vastuussa laitteen viasta vähintään

kahden vuoden ajan, jos valmistaja ei pysty osoittamaan, että vika johtuu laitteen väärästä käsittelystä. Sen sijaan laitteelle myönnetty mahdollinen lisätakuu voi purkautua roottamisen seurauksena, mutta lakisääteinen takuu on silti voimassa. Jos valmistaja ei siis pysty osoittamaan, että ohjelmistoon (engl. *software*) kohdistuva roottaminen on aiheuttanut esimerkiksi laitteen laitteistoon (engl. *hardware*) vian, se kuuluu edelleen takuun piirin. Jos käyttäjä pystyy palauttamaan laitteeseen alkuperäisen ohjelmiston ja poistamaan root-oikeudet (engl. *unrooting*) ja vika jatkuu edelleen, voidaan pitää ilmeisenä, että vika ei johdu roottaamisesta. [46, 47, 48]

Roottaminen antaa paljon vapauksia käyttäjälle mutta samalla myös paljon suuremman vastuun. Laitteen tietoturva on rootatussa laitteessa suuremmassa riskissä. Jos roottaamisesta ei ole enää käyttäjälle enää hyötyä, root-oikeudet voi yleensä poistaa laitteesta.

3.3. Kehittäjäasetuksien käyttäminen

Kehittäjäasetukset eivät ole Android-laitteissa käytössä oletuksena, vaan ne tulee ottaa erikseen käyttöön laitteen asetuksista. Käyttäjän tulee mennä asetusvalikossa laitteen tietoihin ja klikata kohtaa ”ohjelmistoversion numero” seitsemän kertaa. Kehittäjäasetuksissa on runsaasti erilaisia toimintoja ja ne on suunnattu ensisijaisesti sovelluskehittäjille ja edistyneemmille käyttäjille, jotka haluavat muokata Android-laitteen toimintoja normaalia syvemmällä tasolla. Valikkoa ja sen toimintaa esitetään kuvassa 9. Kuvassa vasemmalla on ruutukaappaus kehittäjäasetus-valikon alkunäkymästä, oikealla taas sama valikko, jossa asettelun rajat on asetettu näkyviin. Tällöin näytön elementtien rajat ja marginaalit ovat näkyvillä erivärisinä viivoina ja väreinä.



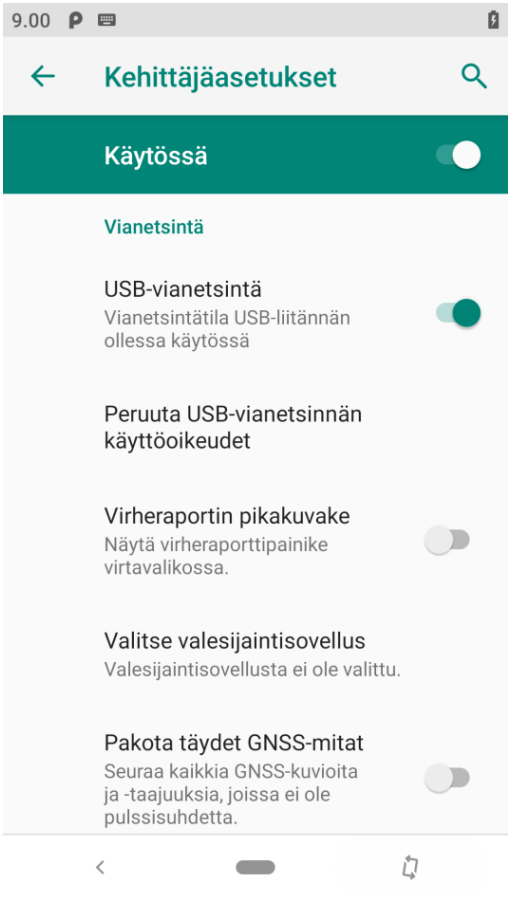
Kuva 9. Kehittäjäasetus-valikko ja esimerkki toiminnasta.

Kehittäjäasetuksien kautta voi tehdä laitteelle suurempia muutoksia kuin normaalien asetusten kautta. Nämä toiminnot voivat vaarantaa myös laitteen tietoturvan. Tällaisia toimintoja ovat esimerkiksi näytön pakottaminen pysymään käynnissä rajattoman pitkän ajan laitteen ollessa latauksessa, OEM-lukituksen kytkeminen pois päältä, jolloin laite on mahdollista rootata ja virheraportin luonti, joka pitää sisällään monenlaista lokitietoa. Lokeista löytyy käyttäjän yksityisiä tietoja, kuten asennetut sovellukset, yhteystietojen nimet sekä numerot, joihin on soitettu ja vastattu viime käynnistyskerran jälkeen. Kehittäjäasetuksista voi myös kytkeä päälle USB-vianetsinnän. Kun laitteen yhdistää tietokoneeseen tämän ominaisuuden ollessa päällä, tietokoneella voi tehdä laitteeseen monenlaisia muutoksia, esimerkiksi asentaa ja poistaa sovelluksia, kirjoittaa, kopioida ja poistaa tiedostoja ja muuttaa laitteen asetuksia. Center for Internet Security -järjestö (CIS) suosittelee julkaisemassaan Androidin turvallisuusarviointipaperillaan pitämään kehittäjäasetuksia pois päältä vedoten USB-vianetsinnän ja muiden kehittäjäasetusten ominaisuuksien tuomaan tietoturvariskiin [49].

Kehittäjäasetukset voi kytkeä pois päältä helposti kehittäjäasetus-valikosta. Valikon yläreunassa on valitsin, joka kytketään pois päältä. Tällöin myös kaikki aiemmin käyttöön otetut kehittäjäasetusten ominaisuudet otetaan pois käytöstä.

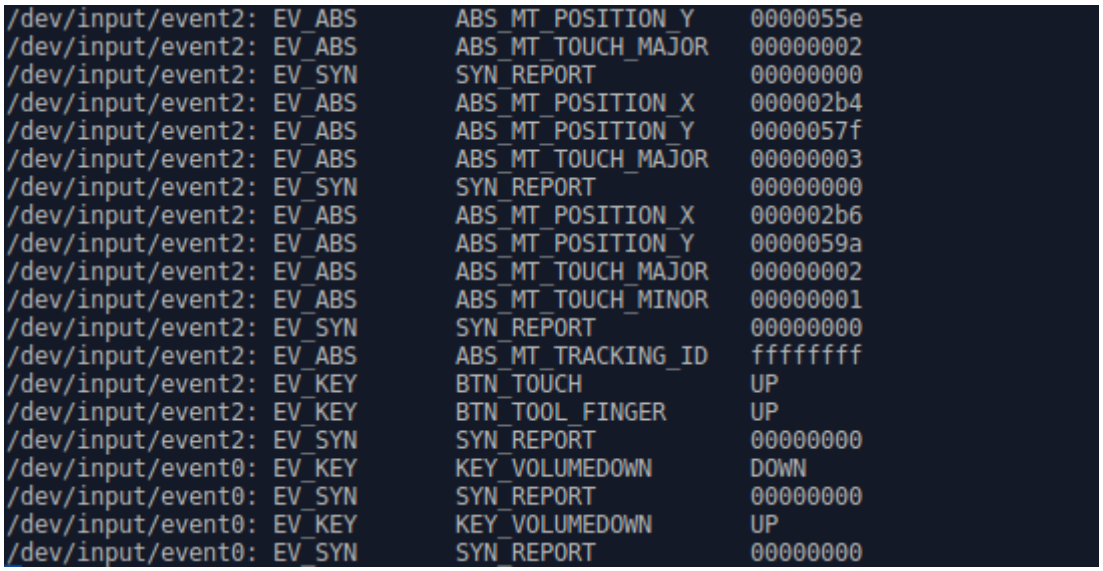
3.4. ADB-yhteyden käyttäminen

ADB (Android Debug Bridge) on monipuolinen komentorivipohjainen työkalu Android-laitteiden kanssa kommunikointiin. Sillä voi muun muassa asentaa ja poistaa debugattavia sovelluksia sekä ajaa Androidin Unix shellin (komentorivitulkin) komentoja, kuten esimerkiksi *ls*, *cat*, *cp* ja *mv*. ADB-yhteys otetaan käyttöön kehittäjäasetusten kautta sallimalla USB-vianetsintä, kuten kuvassa 10 näkyy. ADB-yhteyttä voidaan käyttää USB:n kautta tietokoneelta, kun sille on asennettu ADB-työkalu, joka on vapaasti saatavilla Androidin kehitystyökalujen mukana. ADB-yhteys pitää vielä erikseen hyväksyä Android-laitteella, kun se yhdistetään tietokoneeseen, joka pyytää ADB-yhteyttä laitteeseen. ADB-yhteyden voi määrittää toimimaan myös TCP/IP-protokollan kautta itse määritellyssä portissa. Tällöin siihen voi yhdistää laitteen esimerkiksi WLAN:n kautta, kun Android-laite ja sen ADB-yhteyteen yhdistävä laite ovat samassa verkossa. [50]



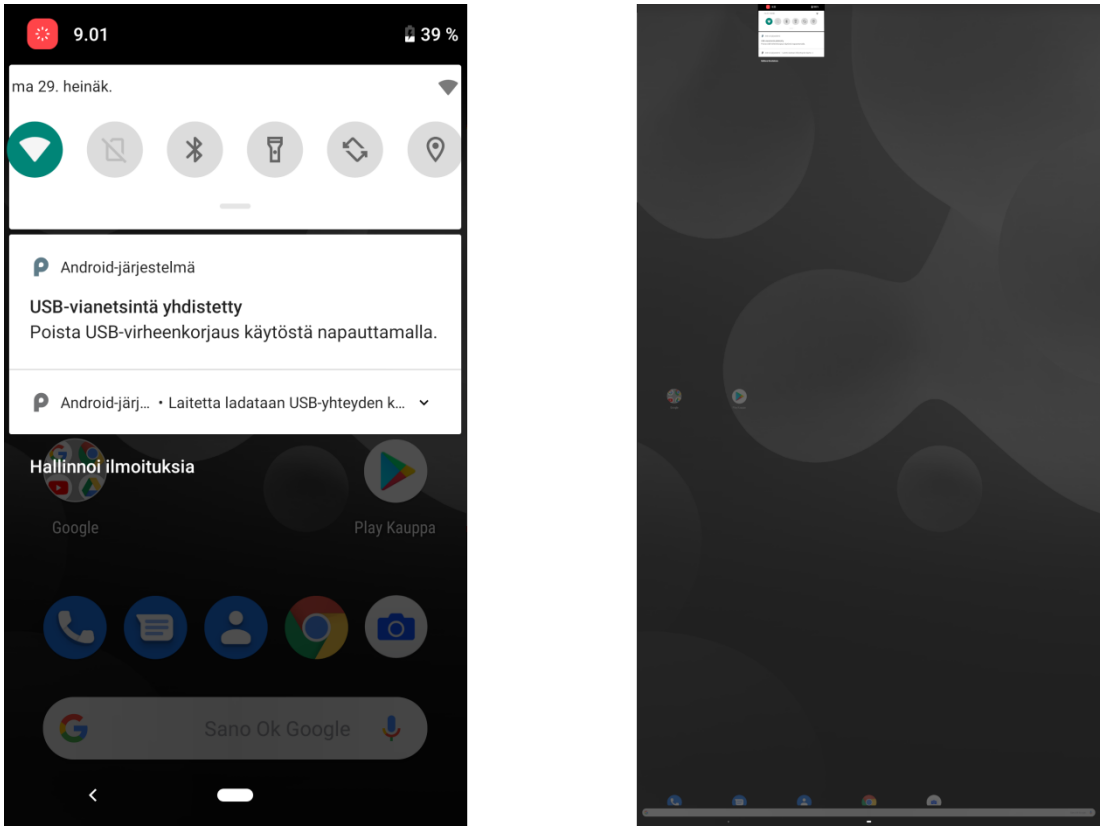
Kuva 10. USB-vianetsinnän salliminen kehittäjäasetuksista ADB-yhteyttä varten.

ADB-yhteyden kautta laitteesta voi ottaa kaikenlaista tietoa, myös yksilöivää ja arkaluonteista sellaista. Esimerkiksi ADB shellin kautta annetulla *dumpsys*-komennolla saa tietoa laitteessa toimivista palveluista. Tämän komennon avulla voi saada tietoon mihin numeroihin puhelimella on soitettu ja vastattu viimeisen käynnistyskerran jälkeen, kuten yllä mainitussa kehittäjäasetusten virheraportissa (virheraportti sisältää *dumpsys*-komennon antamat tiedot). *Dumpsys*-komento ei kuitenkaan anna esimerkiksi saapuneisiin tekstiviesteihin liittyviä puhelinnumeroita tai tekstiviestin sisältöä eli ihan kaikkea tietoa ei ole saatavilla. Kuitenkin yhdistelemällä ADB-yhteyden kautta ajettavia eri komentoja, kuten *dumpsys*, *logcat* ja *getevent*, pystytään laitteen käytöstä päättelemään jo paljon. *Logcat*-komento antaa laitteen systeemilokit [51] ja *getevent*-komento kertoo laitteen syötetapahtumat eli esimerkiksi näppäinpainallukset ja näytön kosketukset ja kosketusten sijainnit näytöllä [52]. Kuvassa 11 on esimerkki *getevent*-komennon tulosteesta. Tämä tuloste saatiin, kun älypuhelimen näytölle piirrettiin ympyrä ja lopuksi painettiin äänenvoimakkuuden pienennysnäppäintä. Näitä lokitietoja (systeemilokit, syötetapahtumaloki, palvelut) yhdistelemällä voi saada selville vaikka pankkisovellukseen kirjatun käyttäjätunnuksen ja salasanan sekä kirjoitettujen viestien sisällön. Tämä vaatii kuitenkin paljon lokien lukutaitoa ja työtä sekä tietenkin ADB-yhteyden laitteeseen.



Kuva 11. Esimerkki ADB shellin kautta annetun *getevent*-komennon tulosteesta.

Hwang, Lee, Kim ja Ryu (2015) esittelivät ADB-yhteyttä vastaan toimivia hyökkäysvektoreita käsittelevässä julkaisussaan [53] muun muassa seuraavanlaiset ADB-yhteyden tuomat riskit ja hyökkäystavat: tekstiviestien lähetys käyttäjältä kysymättä sekä *wm* (windows manager) -työkalulla tehtävä DoS-hyökkäys laitteelle itselleen. DoS-hyökkäys toteutetaan asettamalla ruudun pikselitiheyden liian pieneksi tai suureksi. Kun pikselien määrän asettaa paljon suuremmaksi kuin laitteen näytön pikseleiden fyysisen määrän (esimerkkikomento: ”adb shell wm size 10000x10000”), laite ei enää välttämättä vastaa kosketukseen vaikka laitteen fyysiset napit toimisivat, sillä se ei pysty prosessoimaan kunnolla näytön suurta pikselimäärää tai sen kosketus ei vastaa enää näytön koko alueelta. Tämän komennon tuloksia esitellään kuvassa 12, jossa vasemmalla on kuvakaappaus näytöstä normaalissa tilassa ja oikealla sama tilanne, kun ruudun pikselitiheys on asetettu maksimiinsa. Ilman ADB-yhteyttä laitetta ei voi enää saada toimintakuntoiseksi muuten kuin tehdasasetusten palautuksella, jolloin laitteen tiedot menetetään. Tämä hyökkäys ei kuitenkaan toimi kaikkia Android-laitteita vastaan. Laitteet, joissa ei ole kovin paljon prosessointitehoa, ovat alttiimpia edellä mainitun hyökkäyksen onnistumiselle. ADB-yhteyden kautta voi myös lähettää tekstiviestejä haluttuun numeroon, esimerkiksi maksullisiin viestipalveluihin, ilman että laitteen näytölle tulee mitään ilmoitusta tai vahvistyspyyntöä. Nämä hyökkäykset toimivat testattaessa Android 9 -käyttöjärjestelmällä toimivalla puhelimella. Alkuperäisessä, vuonna 2015 tehdyssä julkaisussa oli mainittu vain Android-versiot 3.0, 4.2.2 ja 4.4 [53].



Kuva 12. Näytön pikselitiheyttä muutettu oikealla niin, että laitetta ei voi käyttää.

ADB-yhteyden kautta tehtävät hyökkäykset vaativat fyysisen pääsyn laitteeseen ja sen näytön lukituskoodin. Android 4.2.2 -päivityksessä tehtiin muutos, jonka myötä ADB-yhteys pitää erikseen hyväksyä laitteelta ja tämä pitää tehdä kun laitteen näytön lukitus on avattu [50]. ADB-yhteys aukaisee paljon väyliä hyökkäyksen tekemiseen myös edellä esitettyjen lisäksi. Myös CIS-järjestö varoittaa Androidin turvallisuusarviointipaperillaan USB-vianetsinnän vaaroista [49]. Tämän takia USB-vianetsintä on hyvä pitää pois päältä Android-laitteista. Se voidaan kytkeä pois joko kehittäjäasetuksista erikseen tai kytkemällä kehittäjäasetukset kokonaan päältä, jolloin kaikki kehittäjäasetukset palautetaan oletusarvoihinsa.

3.5. USB-massamuistin käyttäminen

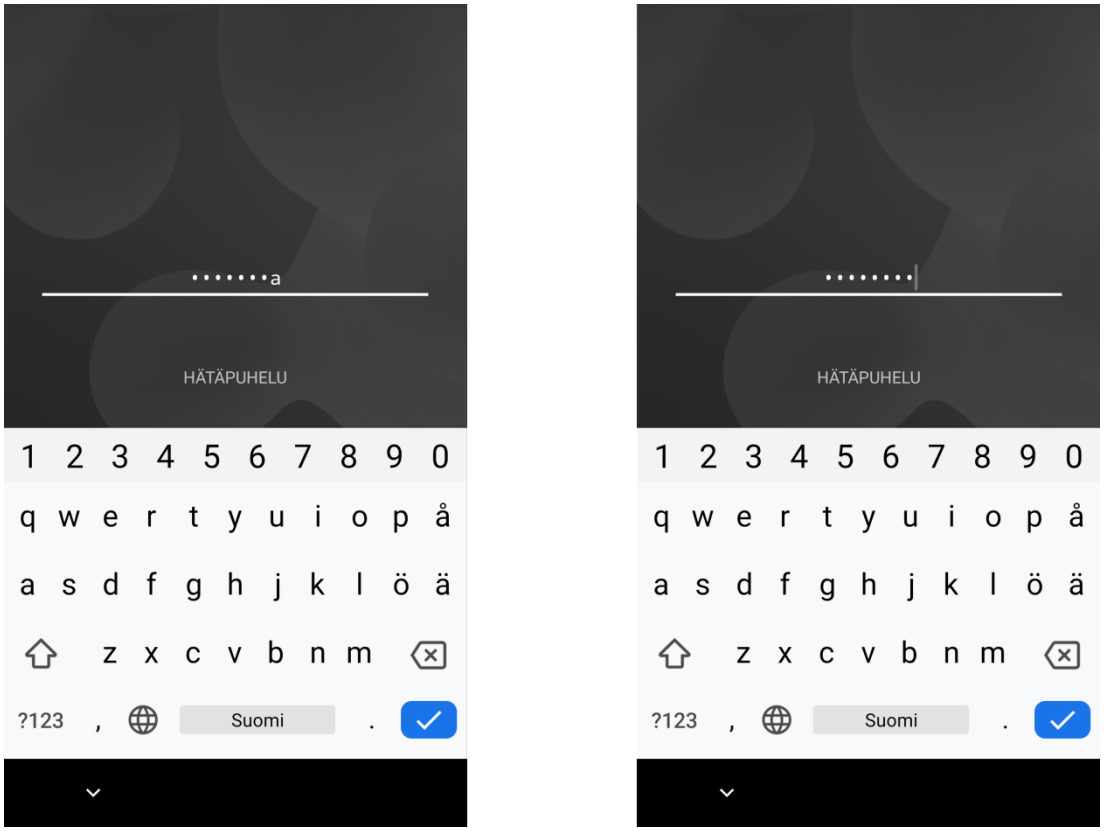
Kun Android-laitteen kytkee tietokoneeseen USB-kaapelilla, tietokoneelle voi antaa luvan tiedonsiirtoon. Tällöin tietokoneelta on pääsy laitteen käyttäjän tiedostoihin, kuten kuviin ja dokumentteihin ja ne voidaan esimerkiksi kopioida laitteelta tietokoneelle tai poistaa laitteelta. Tiedonsiirtoyhteyttä varten laitteen näytön lukitus tulee olla avattuna. Jotkin Android-laitteet tukevat tiedonsiirtoyhteyttä myös toiselta Android-laitteelta, eli puhelimen voi esimerkiksi yhdistää toiseen USB-kaapelilla ja tiedostoja voi muokata tiedonsiirtoluvan saaneella puhelimella.

Jos USB-yhteyttä haluaa käyttää vain latausta varten, tiedonsiirtoa ei tulisi sallia. Tiedostojen siirtoa varten käyttäjän tulisi olla varma siitä, mitä laitteen tiedostoille

tehdään tietokoneen kautta. Yhteyden voi tarvittaessa katkaista joko estämällä tiedonsiirron Android-laitteelta tai irrottamalla USB-johdon.

3.6. Salasanan näkyminen

Android-laitteen suojausasetuksista voi kytkeä salasanojen merkit näkymään hetkellisesti niitä kirjoittaessa. Tällöin salasanoja tai PIN-koodia kirjoittaessa viimeisin kirjoitettu merkki näkyy noin kahden sekunnin ajan. Tämä toiminto on voimassa vain salasanojen tekstinsyöttökenttiin, joissa syötetyt merkit korvataan pisteillä. Salasanan näkymistoiminto on esitetty kuvassa 13. Siinä vasemmalla salasanojen näkyvyys on asetettu päälle, oikealla pois päältä.

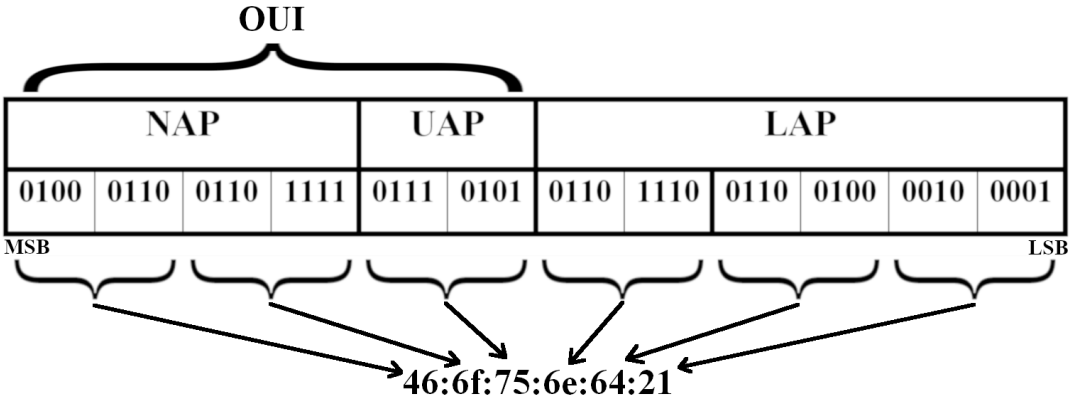


Kuva 13. Salasanan näkyminen näytön lukituksen avaamisessa.

Salasanaa voi olla hankala saada selville, vaikka näkyvyysasetus olisi päällä ja salasanaa selvittävä henkilö katsoisi, kun salasanaa kirjoitetaan. Jos Android-laitteelle sen sijaan on asennettu haittasovellus, joka tallentaa näytön tapahtumat videolle tai ottaa niistä ruutukaappauksia, näistä voi päätellä jo salasanan. Jotkin näppäimistösovellukset saattavat tosin näyttää visuaalisesti, mitä näppäintä kulloinkin painetaan. Tällöin salasanan näkyvyysasetuksen pois kytkemisestä ei ole samanlaista hyötyä.

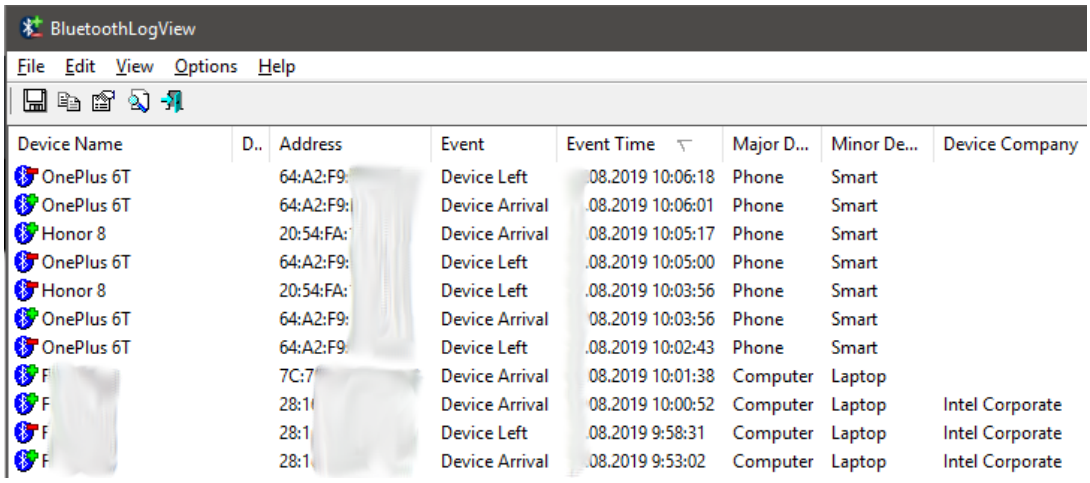
3.7. Bluetoothin löydettävyys

Bluetoothin ollessa päällä Android-laitteella laite voi olla joko löydettävissä tai piilossa muilta Bluetooth-laitteilta. Laitteen tulee olla löydettävissä, jos sen haluaa näkyvän muille laitteille yhdistämistä tai parittamista varten. Jos löydettävyys on päällä, Android-laitetta on mahdollista seurata sen 48-bittisen Bluetooth-osoitteen avulla, joka on yksilöllinen ja pysyvä jokaiselle laitteelle. Siitä on myös mahdollista päätellä laitteen valmistaja. Bluetooth-osoitteen muodostuminen on esitetty kuvassa 14. Osoitteen kolme ensimmäistä tavua eli 24 bittiä, NAP (Non-significant Address Part) ja UAP (Upper Address Part) -osuudet, muodostavat yhdessä OUI-osuuden (Organizationally Unique Identifier) [54, 55]. OUI määräytyy laitteen valmistajan mukaan ja loppuosuus, LAP (Lower Address Part), on valmistajan jokaiselle laitteelle yksilöimä osoiteosuus. Kuvassa vasemmanpuoleisin bitti on MSB (Most Significant Bit) ja oikeanpuoleisin LSB (Least Significant Bit). Osoite esitetään yleensä kuvan alareunassa näkyvässä heksadesimaali-muodossa.



Kuva 14. Bluetooth-osoitteen osat.

Kun Bluetooth-osoite on pysyvä laitteelle ja kun Bluetoothia pidetään löydettävänä, sitä voidaan seurata asentamalla Bluetooth-sensori haluttuun paikkaan. Aina kun löydettävässä tilassa oleva Bluetooth-laite kulkee sensorin ohi, laitteen yksilöllinen osoite näkyy. Näin voitaisiin esimerkiksi seurata, milloin yksittäinen henkilö käy syömässä jossain tietyssä ravintolassa. Seuraamisen lisäksi laitteen valmistaja voidaan päätellä osoitteen avulla. Samoin myös laitteen merkin voi päätellä, jos Bluetoothin kautta näkyvää laitteen nimeä ei ole vaihdettu. Yhdistämättömien Bluetooth-laitteiden osoitteita ei näe roottaamattomalla Android-älypuhelimella eikä Windows 10 -laitteilla niiden normaalista Bluetooth-valikosta. NirSoft on kuitenkin tehnyt Windowsille BluetoothLogView-ohjelman [56], jolla Bluetooth-osoitteet näkyvät ilman yhdistämistä. Ohjelman päänäkymä esitetään kuvassa 15. Tällä ohjelmalla näkee löydettävissä olevien Bluetooth-laitteiden nimen, kuvauksen ja osoitteen. Se tallentaa aikaleimat, jolloin laite tulee näkyville (kuvassa vihreä merkintä) ja milloin se katoaa näkyviltä (kuvassa punainen merkintä). Laitteiden yksilöivät tiedot on sensuroitu kuvasta.



Kuva 15. BluetoothLogView-ohjelman näkymä löydettävässä tilassa käyneistä laitteista.

Android-laitteissa Bluetoothin näkyvyyden kytkemistapa riippuu niiden ohjelmistoratkaisuista. Joissakin laitteissa Bluetooth pitää olla kytkettynä päälle ja tämän lisäksi laite pitää erikseen asettaa löydettäväksi Bluetooth-asetusvalikosta. Näin toimii esimerkiksi tätä varten testatut OnePlus-älypuhelimet, OnePlus 5T (Android 8.1.0, OxygenOS 5.1.7) ja OnePlus 6 (Android 9, OxygenOS 9.0.7). Muissa testatuissa Android-älypuhelimissa (Nokia 8, Bittium Tough Mobile ja Bittium Tough Mobile 2) Bluetooth on näkyvissä silloin, kun Bluetooth on päällä ja Bluetooth-valikko on näytössä etualalla. Erillistä kytkintä näkyvyydelle ei ole. Kun käyttäjä poistuu asetusvalikosta esimerkiksi painamalla kotinäppäintä, Bluetoothin näkyvyys kytkeytyy pois päältä. Samanlainen toimintatapa oli myös testatulla iPhone 6S Plus -älypuhelimella (iOS 12.3.1) ja Fujitsun Lifebook U747 -kannettavalla (Windows 10).

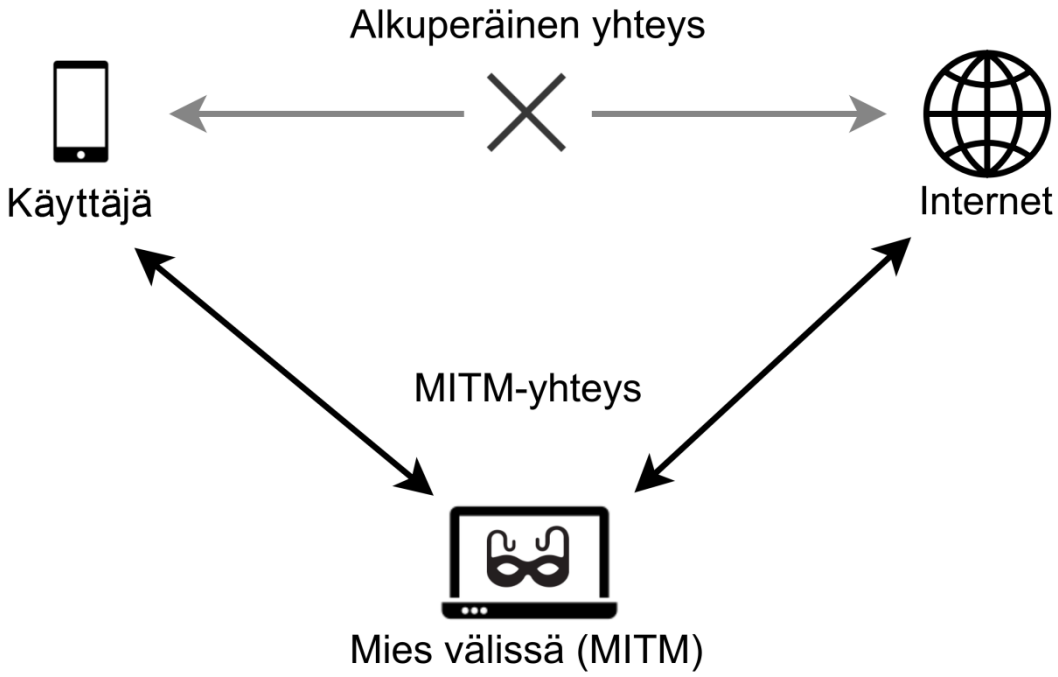
Bluetoothissa on aiemmin todettu monenlaisia tietoturva-aukkoja, esimerkiksi BlueSnarfing, jonka löysi Marcel Holtmann syyskuussa 2003 [57] ja myöhemmin saman vuoden marraskuussa Adam ja Ben Laurie [58]. BlueSnarfing-hyökkäys toimi vanhempia, 2003-vuoden laitteita vastaan, jos ne olivat löydettävässä tilassa tai laitteen Bluetooth-osoite oli tiedossa. Sillä päästiin käsiksi laitteiden tiedostoihin, muun muassa yhteystietoihin, kalenteriin ja IMEI-numeroon [59]. BlueBugging-hyökkäyksessä taas päästiin käsiksi 2014-vuoden Bluetooth-laitteisiin käyttäen hyväksi tietoturva-aukkoa laitteiden ohjelmistossa. Hyökkäyksen avulla saattoi muun muassa kuunnella puheluita, soittaa itse puheluita ja lähettää tekstiviestejä [59, 60]. Vaikka BlueSnarf- ja BlueBug-hyökkäykset eivät toimi moderneja Bluetooth-laitteita vastaan, samanlaisia haavoittuvuuksia voi tulla jatkossakin vastaan. Parhaiten niitä vastaan voi suojautua pitämällä Bluetoothin näkyvyyttä pois päältä, tai kytkemällä Bluetoothin kokonaan pois päältä.

3.8. Avoimien wifi-verkkojen käyttäminen

Tavallisimmat tavat yhdistää Android-laite internetiin ovat matkapuhelinverkko ja wifi. Wifi-verkkojen yhteys voi olla joko suojattu tai suojaamaton. Suojaamatonta wifi-verkkoa kutsutaan avoimeksi. Tällöin verkkoon liittyessä ei tarvitse salasanaa,

vaan kuka tahansa voi liittyä verkkoon. Avoimen wifi-verkon liikenne ei ole salattua ellei yhteys ole erikseen salattu esimerkiksi VPN:llä tai HTTPS-yhteydellä.

Avoimia wifi-verkkoja on tarjolla yleensä julkisilla paikoilla, kuten lentokentillä ja kahviloissa. Oman wifi-verkon eli hotspotin luonti on nykyään hyvin helppoa, se onnistuu jopa Android-laitteella muutamalla näytön painalluksella. Mahdollisen hyökkääjän tai tiedonkalastelijan on siis helppo luoda esimerkiksi kahvilassa oma verkko ja nimetä se houkuttelevasti. Ajatellaan esimerkkiskenaario, jossa kahvila tarjoaa asiakkaidensa käyttöön suojatun wifi-verkon, jonka nimi eli SSID (service set identifier) on ”Coffeehouse Wi-Fi” ja salasanan saa kahvilan henkilökunnalta. Hyökkääjä voi astua kahvilaan, tilata kahvin, istua alas kannettavansa ja Wi-Fi-reitittimensä kanssa, luoda oman, suojaamattoman wifi-verkon reitittimensä avulla ja nimetä sen ”Coffeehouse WLAN”-nimiseksi. Verkon jättäminen avoimeksi houkuttelee tavallisen kahvila-asiakkaan yhdistämään laitteensa hyökkääjän verkkoon sen helppouden takia, jolloin hyökkääjällä on pääsy kaikkeen asiakkaan internet-liikenteeseen ja hän pääse tutkimaan sitä kannettavallaan. Hyökkääjä saavuttaa tällä tavalla niin sanotun ”mies välissä”-aseman (engl. *man-in-the-middle, MITM*), jonka avulla hän voi toteuttaa erilaisia hyökkäyksiä, kuten tunnettujen verkkosivujen vaihtamista huijaussivustoihin (engl. *spoofing*) DNS-kyselyitä (Domain Name System) muokkaamalla [61, 62]. MITM-asemaa on havainnollistettu kuvassa 16, jossa käyttäjä luulee olevansa yhteydessä suoraan Internetiin, mutta todellisuudessa liikenne menee MITM-hyökkääjän kautta.



Kuva 16. MITM-hyökkäys.

Toinen hyökkäystapa, jolla avoimen verkon kautta meneviin tietoihin pääsee käsiksi, on kaapata olemassa olevan avoimen wifi-verkon liikenne luomatta itse uutta verkkoa. Koska avoimen wifi-verkon liikenne kulkee ilmassa radioaaltomuodossa, kuka tahansa voi kuunnella niitä sopivalla vastaanottimella. Tätä varten on olemassa ohjelmistotyökaluja, kuten airodump-ng [63], joka toimii Linuxilla ja Windowsilla.

Tämän ohjelman avulla esimerkiksi Linux-kannettavalla, jossa on USB-porttiin kytketty wifi-vastaanotin, voi kuunnella haluamaansa wifi-verkkoa etsimällä ensin oikean wifi-kanavan ja ottaa sitten sen liikenteestä vedoksen (engl. *dump*). Tätä vedosta voi sitten myöhemmin tutkia tarkemmin ja etsiä siitä esimerkiksi käyttäjätunnuksia ja salasanoja sekä selvittää, millä sivuilla wifi-verkon käyttäjät kävivät. [64]

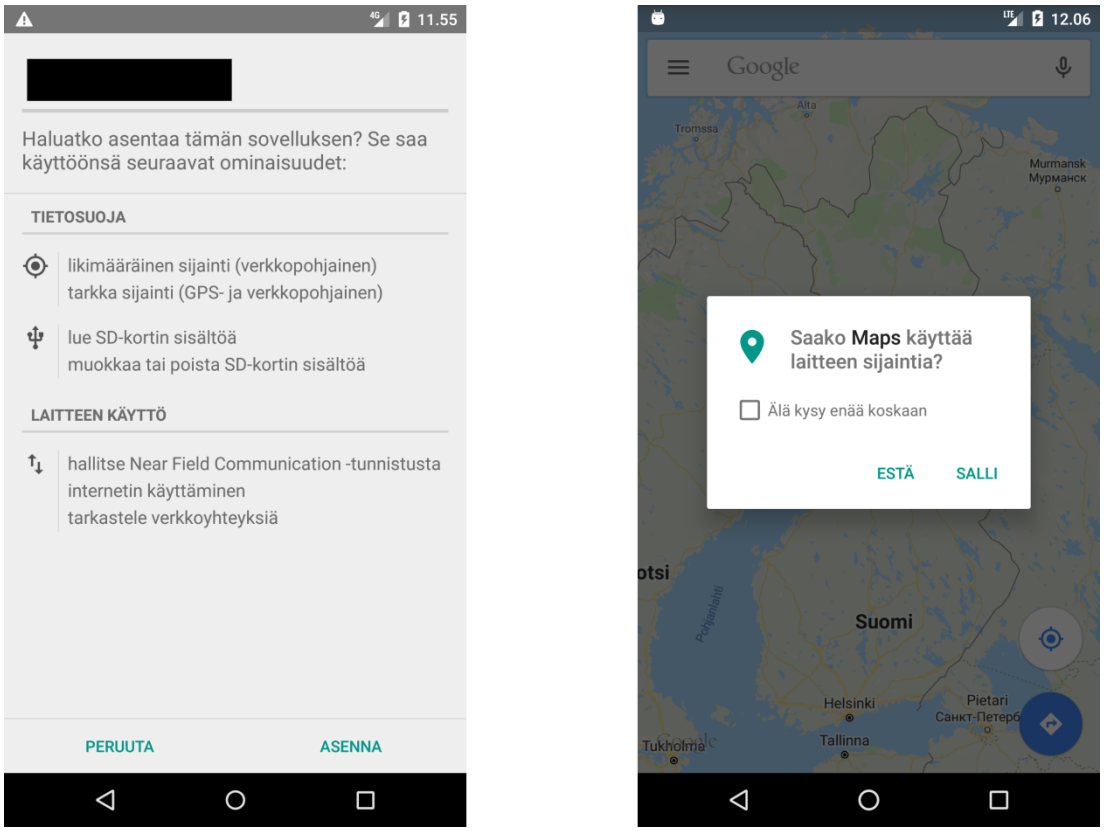
Avoimessa verkossa ollessa voi kuitenkin suojautua edellä mainituilta hyökkäyksiltä käyttämällä esimerkiksi VPN-yhteyttä tai vain sivuja, jotka käyttävät HTTPS-yhteyttä HTTP-yhteyden sijaan [64, 65]. HTTPS-yhteyttä vastaan voi tosin myös tehdä erilaisia MITM-hyökkäyksiä [66], mutta ne ovat paljon työläämpiä ja teknisesti haastavampia kuin salaamatonta HTTP-yhteyttä vastaan tehtävät hyökkäykset. Varminta on kuitenkin yhdistää salausta käyttävään ja luotettavaan wifi-verkkoon. Jos sellaista ei ole käytössä, kannattaa käyttää matkapuhelinverkkoa internet-yhteyteen ennemmin kuin avoimia wifi-verkkoja, sillä matkapuhelinverkon kuunteluun ja sen liikenteen sieppaamiseen tarkoitetut laitteet ovat harvinaisempia ja kalliimpia kuin vastaavat wifi-liikenteelle tarvittavat laitteet. Avoimen wifi-verkon kuunteluun ja sen liikenteen sieppaamiseen soveltuvat jopa ihan tavalliset wifi-sovittimella varustetut kannettavat tietokoneet.

3.9. Sijainnin käyttäminen

Sijaintia voi tarvita Android-laitteilla monenlaiseen tarkoitukseen; sen avulla voidaan navigoida autolla, etsiä lähistöltä kauppoja ja tapahtumia [67], etsiä ystäviä näiden jakamien sijainnin perusteella tai jakaa omaa sijaintia [68], seurata juoksulenkin reittiä ja nopeuksia [69], geokätköillä [70] ja pelata erilaisia sijaintia käyttäviä pelejä, kuten Pokémon Go:ta [71]. Kun sovellus on kysynyt luvan sijainnin käyttöön ja sijainti on kytketty päälle, se voi saada laitteen sijaintitiedon. Sijaintitieto voi olla joko karkea tai tarkka sijainti. Karkea sijainti lasketaan matkapuhelinverkkojen tukiasemien ja saatavilla olevien langattomien verkkojen perusteella [72]. Tarkkaan sijaintiin käytetään laitteen GPS-moduulia, mutta myös karkean sijainnin laskentaa voidaan käyttää tarkan sijainnin laskemisen apuna [72].

Sijainnin käyttöä varten sovellusten tulee esittää vaatimus sijainnin käyttöön manifestissaan. Sovellus voi pyytää manifestissaan joko likimääräistä tai tarkkaa sijaintia. Android 6.0 -versiosta lähtien otettiin käyttöön erillinen käyttöoikeuden kysyminen sijainnille. Aiemmissa Android-versioissa käyttöoikeus sijaintiin kysyttiin sovellusta asentaessa, Android 6.0:sta eteenpäin käyttöoikeus kysytään sovellusta käytettäessä silloin, kun sovellus sitä ensimmäistä kertaa tarvitsee. Tämä teki käyttöoikeuden tarvitsemisesta käyttäjälle helpommin ymmärrettävän ja kontrolloitavan. Samalla haitallisten sovellusten on hankalampi saada käyttöoikeutta, kun sitä ei voida kysyä samalla kertaa kaikkien muiden oikeuksien kanssa, joiden sekaan sijainnin kysely saattoi hautauta. [72]

Käyttöoikeuksien kysymistä esitetään kuvassa 17. Kuvassa vasemmalla on kuvakaappaus sijaintia käyttävän sovelluksen asentamisesta Android 5.1 -käyttöjärjestelmäversiossa (sovelluksen nimi peitetty), oikealla taas on kuvakaappaus Google Maps -sovelluksesta, kun se pyytää sijainnin käyttöoikeutta Android 6.0 -käyttöjärjestelmäversiossa. Annetun käyttöoikeuden voi myöhemmin käydä perumassa sovelluksen asetuksista [73].



Kuva 17. Sijainnin käyttöoikeuden kysymisen ero Android 5.1 ja 6.0 -versioiden välillä.

Sen jälkeen kun sovellus on saanut käyttöoikeuden sijaintiin, se voi tallentaa sijaintitiedon ja lähettää sen myös eteenpäin internetin kautta sovelluksen määrittelemälle palvelimelle. Sovellukset eivät tarvitse internetin käyttöön erillistä käyttöoikeutta, vaan se on niin sanottu ”normaali käyttöoikeus” (engl. *normal permission*), kun taas sijainnin käyttö on ”vaarallinen käyttöoikeus” (engl. *dangerous permission*), johon tarvitaan lupa käyttäjältä [73]. Käyttöoikeuden annettuaan käyttäjän voi olla hankala tai jopa mahdoton hallinnoida sijaintitietonsa käyttöä, jos sovellus ei itsessään tarjoa siihen mitään mahdollisuuksia. Tätä ongelmaa on Euroopan unionissa pyritty korjaamaan lainsäädännöllä. Jos sijaintitiedot on mahdollista yhdistää käyttäjään niin, että käyttäjän pystyy tunnistamaan suoraan tai epäsuorasti, sijaintitiedon kerääjää sitoo EU-maissa GDPR-asetus (General Data Protection Regulation) [74]. Tämä tarkoittaa, että EU-maissa olevien käyttäjien oikeuksia on suojattu niin, että tietoja kerätessä käyttäjältä pitää kysyä ensin suostumus tietojen keräykseen. Käyttäjä voi myös pyytää kerättyjä tietoja tarkasteltavaksi sovelluksen tarjoajalta. Tarvittaessa tiedot voi myös pyytää poistamaan. Jos sovelluksen tekijä rikkoo tätä lakia, siitä voi seurata sakko, joka voi olla kooltaan enintään 20 miljoonaa euroa. Jos kyseessä on yritys, sakko voi olla suurempikin. Yrityksille sakko voi olla myös 4 prosenttia yrityksen edeltävän tilikauden vuotuisesta maailmanlaajuisesta liikevaihdosta, jos sakko tällä tavalla laskettuna ylittää edellä mainitun 20 miljoonan euron rajan [74].

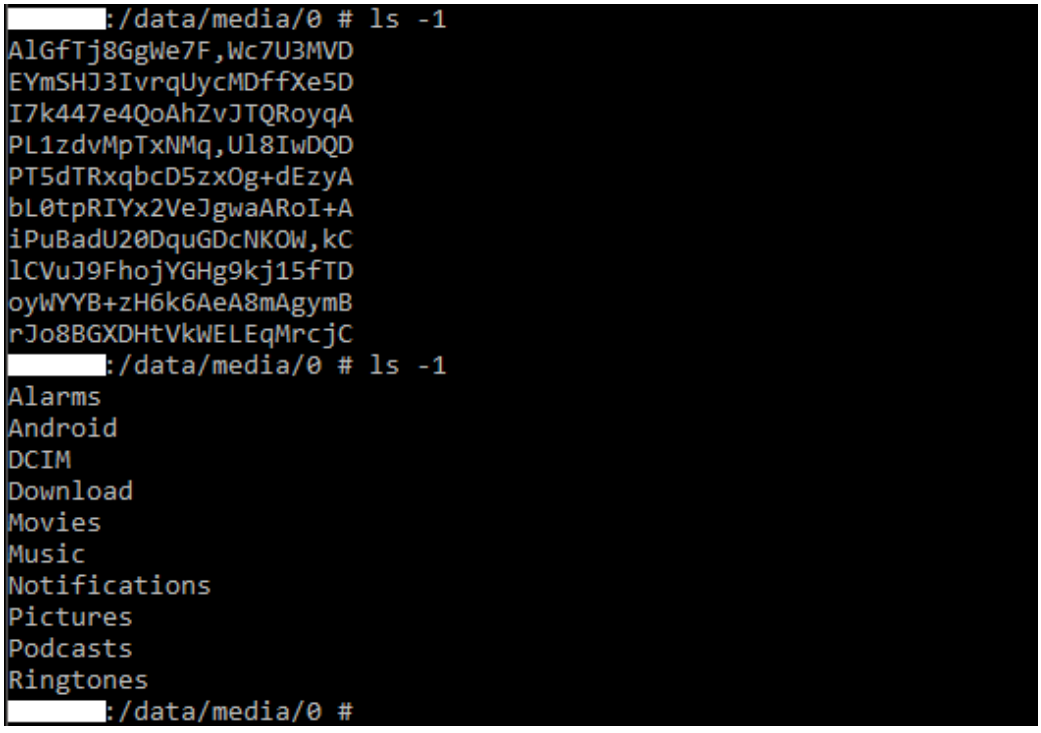
Vaikka EU:ssa laki on yksilön puolella sijaintitietojen yksityisyyden kannalta, sijaintia ei kannata pitää tarpeettomasti päällä sen luoman yksityisyysuhan takia. Jos sovelluksella ei ole selkeää perustelua tai tarvetta sijainnin käytölle, sitä ei kannata

sovellukselle antaa. Jo myönnetyn käyttöoikeuden voi käydä jälkikäteen perumassa sovellukselta tai sovelluksen voi myös kokonaan poistaa.

3.10. Tallennustilan salaaminen

Android-laitteen tallennustila voi olla salaamaton tai salattu. Salaamattomassa tallennustilassa Android-käyttöjärjestelmä ei tee mitään salausta laitteen tiedostoille. Jos laitteen tallennustila salataan, se enkoodataan AES-lohkosalausmenetelmällä (engl. *Advanced Encryption Standard, AES*), joka käyttää symmetristä avainta [23, 24, 75]. Tämä tarkoittaa, että tallennustilan sisältö muutetaan salaamalla sellaiseen muotoon, että sen sisältöä ei voida ymmärtää tai lukea (engl. *scramble*). Näin tiedot ovat tallessa vaikka tallennustilan sisältö onnistuttaisiin sieppaamaan. Tiedot saa takaisin luettavaan muotoon, jos tiedetään salaustapa ja salaukseen käytetty avain [76]. Salausta käyttävillä Android-laitteilla tiedot pysyvät salattuna laitteen ollessa sammutettuna ja tiedostojen salaus puretaan käyttöä varten vasta kun laite on käynnistetty ja laitteen pääsykoodi on syötetty. Salattua ja salaamatonta tallennustilaa voi päästä tutkimaan ohjelmallisesti suoraan laitteelta esimerkiksi USB-yhteydellä tai tekemällä muutoksia itse laitteistoon fyysisesti, esimerkiksi irrottamalla muistipiirin ja tutkimalla sitä.

Kuvassa 18 esitetään Android-laitteen media-kansion sisältöä ADB shellin kautta, kun laitteessa on otettu käyttöön tiedostopohjainen salaus (FBE) ja tiedostojärjestelmän metadatan salaus, eli esimerkiksi kansioiden nimet ja tiedostojen koot on salattu [25]. Salattu sisältö näkyy kuvakaappauksen alkuosassa. Kuvakaappauksen keskivaiheilla käytetään samaa ”ls -1”-komentoa sen jälkeen kun salaus on purettu avaamalla laitteen lukitus, jolloin kansioiden nimet näkyvät salaamattomina.



Kuva 18. Salattu ja salaamaton kansion sisältö tiedostopohjaisella ja metadatan salauksella.

Android-laitteet tukevat 7.0 -käyttöjärjestelmäversiosta eteenpäin tiedostopohjaista salausta, jolla voidaan salata eri tiedostot eri avaimilla [24]. Tätä ennen oli mahdollista käyttää vain koko levyn salausta (FDE). Siinä laitteen käyttäjän kaikki tiedostot salataan yhdellä avaimella [23]. Tiedostopohjaisessa salauksessa on se etu, että se tukee Androidin ”suora käynnistys”-ohjelmointirajapintaa (engl. *Direct Boot API*) [24]. Sen avulla Android-laitteet voivat käynnistyä suoraan näytön lukitusruutuun. Koko levyn salauksessa käyttäjien tuli antaa pääsykoodi ennen kuin mihinkään tietoihin pääsi käsiksi. Tällöin laitteessa eivät toimi muut kuin perustavanlaatuisimmat ominaisuudet. Esimerkiksi käyttäjille tärkeät hälytykset, helppokäyttötoiminnot ja puhelutoiminnot hätäpuheluja lukuun ottamatta eivät toimi. Uudemman, tiedostopohjaisen salaustavan ja ”suora käynnistys”-ohjelmointirajapinnan avulla Androidille on mahdollista tehdä sovelluksia, jotka voivat käynnistyä ja toimia rajoitetusti jo ennen kuin laitteen lukitus (näytön lukitus) on avattu uudelleenkäynnistyksen jälkeen. Sovelluksella tulee tällöin olla manifestissaan merkittynä tietoisuus suorasta käynnistyksestä (engl. *Direct Boot aware*) [24]. Tällaiset sovellukset voivat siis käynnistyä ja toimia määritellyiltä osiltaan ennen kuin käyttäjä on avannut laitteen lukituksen käynnistyksen jälkeen. Näin käyttäjän tiedot ovat turvassa mutta oleelliset toiminnot, kuten puhelutoiminnot ja näppäimistö pääsykoodin syöttöä varten ovat jo päällä ennen kuin laitteen lukitus on avattu.

Salatun tallennustilan purkaminen takaisin luettavaan muotoon ilman salausavainta eli salauksen murtaminen on todella hankalaa, jos salaus on toteutettu AES-salauksella kuten Androidissa. Android käyttää tallennustilan salaukseen 128 tai 256 bitin pituista avainta koko levyn salausta käytettäessä (Android 4.4–6.0) ja 256 bitin pituista avainta kun käytössä on tiedostopohjainen salaus (Android 7.0:sta eteenpäin) [23, 24]. AES-256:ta eli 256 bitin pituista avainta käyttävässä AES-salauksessa mahdollisia avaimia on 2^{256} eli noin $1{,}2 * 10^{77}$ kappaletta. Salauksen voi yrittää murtaa kokeilemalla järjestelmällisesti eri avaimia niin kauan että oikea avain löytyy. Tällaisella raakaan laskentatehoon (engl. *brute force*) perustuvalla niin kutsutulla väsytyshyökkäyksellä ei käytännössä voi nykytekniikalla saada auki AES-salattua tallennustilaa sillä mahdollisten avainten määrä on liian suuri saatavilla olevaan laskentatehoon nähden.

Jos oletetaan modernin tietokoneen kokeilevan 10^9 eli miljardi avainta sekunnissa [77] ja supertietokoneen kokeilevan noin 10^{13} avainta sekunnissa [78], veisi heikomman AES-128 -salauksen murtaminen niin kauan että sitä ei ole järkevää yrittää [79, s. 134-135]. Supertietokoneella kestäisi $5{,}4 * 10^{17}$ vuotta löytää oikea avain olettaen, että avain löytyy, kun puolet mahdollisista avaimista on kokeiltu. Maailmankaikkeuden on laskettu olevan nyt noin 13,8 miljardia vuotta vanha [80], joten tarvittaisiin vielä noin 39 000 000 kertaa tuo aika, että saataisiin murrettua yhdellä supertietokoneella AES-128 -salauksella suojattu tallennustila. AES-128 - ja AES-256 -salauksien murtamiseen tarvittavia aikoja väsytyshyökkäyksellä on esitetty taulukossa 1.

Taulukko 1. Väsytyshyökkäyksen vaatima keskimääräinen aika salauksen purkuun.

Avaimen koko	Avainten määrä	Aikavaatimus 10^9 avaimen kokeilulla/s	Aikavaatimus 10^{13} avaimen kokeilulla/s
128	$2^{128} \approx 3{,}4 * 10^{38}$	$5{,}4 * 10^{21}$ vuotta	$5{,}4 * 10^{17}$ vuotta
256	$2^{256} \approx 1{,}2 * 10^{77}$	$1{,}8 * 10^{60}$ vuotta	$1{,}8 * 10^{56}$ vuotta

Tavallisen väsytyshyökkäyksen lisäksi AES-salausta on yritetty murtaa monella muulla tavalla, joista kehittynein laskentaan perustuva hyökkäys on niin sanottu täydellisen kaksijakoisen verkon hyökkäys (engl. *biclique attack*) [81]. Sen avulla voidaan vähentää salauksen purkamiseen tarvittavan laskennan kompleksisuutta (esimerkiksi AES-128:ssa tarvittavan laskennan määrä laskee luvusta 2^{128} lukuun $2^{126,18}$), mutta tämänkin hyökkäyksen avulla saaduista eduista huolimatta jäävät yleisesti käytössä olevat AES-128 ja sitä vahvemmat AES-salaustavat edelleen käytännössä mahdottomiksi murtaa. Taulukkoon 2 on laskettu tämän hyökkäyksen aikavaatimukset oletuksella, että avain löytyy kun puolet avaimista on kokeiltu.

Taulukko 2. Salauksen purku täydellisen kaksijakoisen verkon hyökkäyksellä.

Avaimen koko	Avainten määrä	Aikavaatimus 10^9 avaimen kokeilulla/s	Aikavaatimus 10^{13} avaimen kokeilulla/s
128	$2^{126,18} \approx 9{,}6 * 10^{37}$	$1{,}5 * 10^{21}$ vuotta	$1{,}5 * 10^{17}$ vuotta
256	$2^{254,42} \approx 3{,}8 * 10^{76}$	$6{,}1 * 10^{59}$ vuotta	$6{,}1 * 10^{55}$ vuotta

Salausta voidaan yrittää purkaa perinteisen laskennallisen tavan lisäksi myös sivukanavahyökkäyksillä (engl. *side-channel attacks*). Ne eivät ole niinkään riippuvaisia salaustavan matemaattisesta toteutuksesta. Sivukanavahyökkäyksellä yritetään saada salausavain selville käyttäen hyväksi salauksen toteuttamiseen käytetyn laitteiston tai ohjelmiston suunnitteluvirheitä tai haavoittuvuuksia, jotka perustuvat yleensä virrankulutukseen tai ajallisiin tietoihin (engl. *timing*) [82]. Esimerkki sivukanavahyökkäyksien paremmasta tehosta perinteisiin laskemalla tapahtuviin murtamistapoihin verrattuna on Ashokkumarin, Girin ja Menezesin (2016) esittelemä hyökkäystapa, jonka avulla onnistuttiin tietyin edellytyksin murtamaan AES-128 salaus alle minuutissa [83]. Tämä hyökkäys perustuu välimuistiin tehtävien pyyntöjen hyväksikäyttämiseen (engl. *cache access attack*).

Uhka sille, että salaamattomaan tallennustilaan päästään käsiksi, on pienempi kuin esimerkiksi laitteen verkkoliikenteen kuunteleminen wifi-verkossa. Wifi-verkon kuunteleminen on teknisesti helpompaa ja se on myös huomaamattomampaa, sillä se ei vaadi fyysistä pääsyä laitteeseen. Jos laitteeseen on fyysinen pääsy, ilmeisin tapa päästä tallennustilaan käsiksi olisi sen USB-portin kautta. Jos laite on suojattu pääsykoodilla, sen tietoihin ei yleensä päästä käsiksi, sillä USB-tiedonsiirtoyhteys pitäisi ensin sallia laitteella. Tämä onnistuu vasta kun laitteen näytön lukitus on avattu.

Tietokoneissa, joissa on irrotettava kovalevy, on helpompi päästä käsiksi salaamattomiin tietoihin. Tämä vaatii vain kovalevyn irrottamisen ja kytkemisen toiseen tietokoneeseen, jolla voi selata irrotetun kovalevyn sisältöä suoraan. Android-laitteissa, kuten puhelimissa ja tableteissa, on kuitenkin pääsääntöisesti kiintolevyn sijaan Flash-muisti. Se on kolvattu suoraan emolevyyn eikä siten ole irrotettavissa ilman oikeita työkaluja (kuumailma-asema, emolevyn pidiketeline eli jigi sekä pinsetit) ja osaamista (laitteen purkaminen emolevyn irrotusta varten, oikeat lämpötilat kuumailma-asemalle, muiden komponenttien suojaus tarvittaessa, tarkkuus pinsettien käytössä) [84]. Irrotuksen jälkeen muistipiirin liittimien reititys (engl. *pin-out* tai *pinout*) tulee selvittää ja kolvata muistipiiriä lukeva laite muistipiirin oikeisiin liittimiin [84]. Piirin irrottamiseen ja lukemiseen tarvittavat työkalut eivät ole kalliita hankkia, mutta tarvittava osaaminen, irrotusprosessi, muistipiirin lukeminen ja aiheeseen perehtyminen tekevät tällä tavalla tapahtuvasta tietojen lukemisesta varsin hankalan prosessin.

Flash-muistin lukemiseen suoraan emolevyltä on tehty myös työkaluja, joiden avulla muistipiiriä ei tarvitse irrottaa. Nämä työkalut hyödyntävät erilaisia emolevyssä olevia diagnostiikka- ja testausliitäntöjä, kuten JTAG (Joint Test Action Group) ja eMMC ISP (embedded MultiMediaCard In-System Programming). Näitä liitäntöjä käytetään yleensä laitteiden testaamisessa niiden tuotantovaiheessa. Liitännät voidaan piilottaa tai poistaa emolevyistä siinä vaiheessa, kun laitteet siirtyvät lopputuotantoon. Ne voidaan myös poistaa käytöstä ohjelmallisesti, jolloin liitännöille ei ole enää rajapintaa ohjelmiston puolelta lopputuotteessa. Emolevylle piilotetut liitännät on mahdollista selvittää, mutta se on työlästä ja vaatii erityisosaamista. [85]

Jos Android-laitteessa on SD-muistikortti (yleensä microSD-kortti), jolle on tallennettu kuvia tai muita tiedostoja eikä korttia ole suojattu salauksella, tietojen luvaton lukeminen on vaivatonta. Tällöin prosessi on sama kuin tietokoneen kiintolevyn lukemisessa, eli SD-kortti irrotetaan Android-laitteesta ja asetetaan toiseen laitteeseen, jolla korttia voi lukea. Android tukee versiosta 9 ylöspäin sekä SD-kortin ja laitteen oman sisäisen muistin yhtäaikaista salausta. Android 7.0–8.1 -versioissa laitteen tiedostopohjaista salausta ei voinut käyttää yhtä aikaa muistikortin salauksen kanssa. Jos 7.0–8.1 -versiolla varustetussa laitteessa käyttää tiedostopohjaista salausta laitteen muistille, muistikorttia ei voi salata, vaan sitä täytyy käyttää tavallisena, siirrettävänä muistina. [24]

Android-laitteen voi salata sen suojausasetuksien kautta, jos laite tukee sitä. Laitteen salauksen tila (salattu tai salaamaton) näkyy myös samassa asetusvalikossa. Monet uudet Android-laitteet salaavat tallennustilansa automaattisesti jo niiden käyttöönoton yhteydessä.

3.11. Tuntemattomat sovelluslähteet

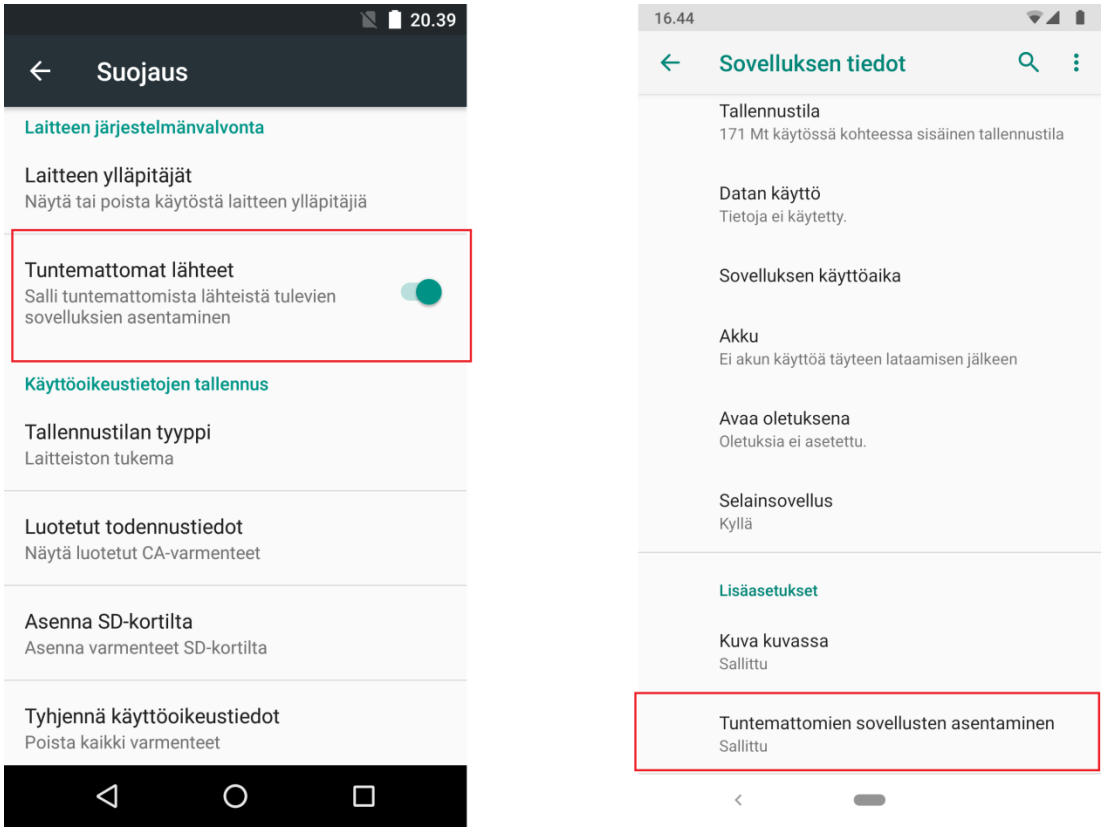
Android-laitteissa tulee oletuksena Googlen oma sovelluskauppa Google Play, jos laite on läpäissyt Googlen asettamat Android-yhteensopivuustestit ja -vaatimukset sekä laitteen valmistaja on ostanut Googlen mobiilipalveluiden (Google Mobile Services, GMS) lisenssin [86]. Google tarkistaa Play-sovelluskauppaansa ladatut sovellukset osana Google Play Protect -järjestelmäänsä [87]. Myös muista sovelluslähteistä ladatut sovellukset tarkistetaan oletusarvoisesti Play Protect -järjestelmällä ja tuntemattomat sovellukset lähetetään Googlelle analysointia varten. Käyttäjä voi kuitenkin kytkeä nämä ominaisuudet pois päältä turvallisuusasetuksista.

Tuntemattomat sovelluslähteet tulee sallia, jos haluaa asentaa sovelluksia, jotka on ladattu muualta kuin Play-kaupasta. Tämä altistaa siis käyttäjän mahdollisesti tarkastamattomille sovelluksille. Käyttäjällä ei ole samanlaista varmuutta siitä, onko sovelluksessa mahdollisesti selvästi haitallisia ominaisuuksia, jotka voivat esimerkiksi urkkia käyttäjän tietoja tai näyttää mainoksia häiritsevällä tavalla. Kaikki muut lähteet paitsi Play-kauppa luokitellaan tuntemattomiksi sovelluslähteiksi. Näitä lähteitä voivat olla esimerkiksi internet-selaimet sekä muut sovelluskaupat, kuten avoimen lähdekoodin sovelluksia tarjoava F-Droid [88].

Sovelluksille, jotka on ladattu muualta kuin Play-kaupasta, ei myöskään tarjota automaattisesti päivityksiä, ellei niitä ole ladattu sellaisesta sovelluskaupasta, joka tukee sovellusten päivittämistä. Käyttäjä ei esimerkiksi saa tietoa selaimen kautta ladattujen sovellusten mahdollisista uusista versioista ellei ota niistä itse selvää. Vanhentuneissa, päivittämättömissä sovelluksissa voi olla tietoturva-aukkoja, jotka sovelluksen kehittäjä on voinut korjata uudemmissa versioissa. Sovelluspäivitysten

tarjoamisen automaattinen puuttuminen voidaan myös siis katsoa olevan tietoturvariski.

Tuntemattomat sovelluslähteet sallitaan ja kytketään pois päältä Android 6–7.1.1 -versioissa kokonaisvaltaisesti kaikille mahdollisille lähteille yhdestä paikasta suojausasetuksissa. Tätä esitellään kuvan 19 vasemmanpuoleisessa kuvakaappauksessa. Android 8.0 -versiosta lähtien tämä asetus muuttui sovelluskohtaiseksi [89]. Asetus pitää käydä erikseen kytkemässä kyseisen sovelluksen, esimerkiksi internet-selaimen, sovellusasetuksista. Tätä esitellään kuvan 18 oikeanpuoleisessa kuvakaappauksessa. Tästä 8.0-versiossa tehdystä muutoksesta johtuen tuntemattomien sovelluslähteiden tilaa ei pystytä tavallisen sovellusten ohjelmointirajapinnan kautta enää määrittämään, vaan sen arvo jätetään tässä työssä tehdyssä sovelluksessa tyhjäksi (N/A, *not available*) Android-version ollessa 7.0 tai suurempi.



Kuva 19. Tuntemattomien sovelluslähteiden salliminen Android 6 - ja 9 -versioissa.

Google on saanut myös kritiikkiä Play-kauppansa sovellusten tarkistusprosessista. Sovelluskauppaan on päätynyt ajan saatossa monenlaisia haitallisia sovelluksia, jotka ovat päässeet tästä prosessista läpi [90]. Yksi esimerkki tällaisista ovat sovellukset, jotka näyttävät mainoksia jopa Android-laitteen kotinäytöllä ja piilottavat itsensä sovellusvalikosta, mikä tekee niiden havaitsemisesta ja yhdistämisestä mainoksiin hankalaa [91]. Google onkin yrittänyt parantaa sovellusten tarkistusprosessiaan. Marraskuussa 2019 Google kertoi aloittavansa yhteistyön ESET-, Lookout- ja Zimperium-tietoturvayhtiöiden kanssa [92]. Yhtiöt muodostivat yhdessä Googlen kanssa App Defence Alliance -liittouman ja yhtiöiden skannausmoottorit yhdistetään osaksi Google Play Protect -järjestelmää.

3.12. Muita osatekijöitä

Edellä mainittujen tietoturvan osatekijöiden lisäksi on paljon muitakin osatekijöitä, joihin olisi pääsy sovellusten ohjelmointirajapinnan kautta. Paljon on myös sellaisia osatekijöitä, joihin ei tämän rajapinnan kautta tavallisella sovelluksella pääse käsiksi, mutta joiden voidaan katsoa olevan myös olennaisia asioita tietoturvan kannalta. Joihinkin laitteen tietoihin pääseminen vaatisi siis järjestelmätason oikeudet, mutta tässä työssä keskityttiin tavallisen sovelluksen saatavissa oleviin tietoihin.

Tavallisten sovellusten ohjelmointirajapinnalla voisi tarkistaa tässä luvussa aiemmin mainittujen 11 osatekijän lisäksi esimerkiksi Android-version (uusimmissa versioissa tietoturvaan liittyviä parannuksia), tietoturvapäivitysten päivämäärän (käyttöjärjestelmäversiosta erillinen päivitys) ja OEM-lukituksen päälläolon eli käynnistyslataajan lukituksen (engl. *bootloader lock*) tilan. Laitteesta voi tarkistaa myös ovatko siinä käytössä laitteistopohjainen avainvarasto ja VPN-yhteys.

Uusien tietoturvan osatekijöiden selvittämistyö ja tarkistamisen varsinainen toteuttaminen vaativat kuitenkin aikaa, osa enemmän ja osa vähemmän. Työn rajoittamiseksi sovellukseen toteutettavien tarkistettavien osatekijöiden määrä jätettiin kohtuulliseksi, sillä tarkoitus oli pikemminkin saada konseptin kokeiluun (engl. *proof of concept*) sopiva sovellus. Sovelluksen kehitystä jatkettiinkin edelleen tämän työn jälkeen.

Tutkittujen ja toteutettujen osatekijöiden tarkistusten lisäksi oli runsaasti osatekijöitä, joita ei voitu tavallisella sovelluksella testata Androidin hiekkalaatikoinnista johtuen tai joiden tarkistukseen ei ole olemassa virallista rajapintaa. Järjestelmätason sovelluksella ainakin osa näistä olisi mahdollista toteuttaa. Tällaisia osatekijöitä ovat esimerkiksi SIM-kortin PIN-lukituksen ja Smart Lock -ominaisuuden [93] käyttäminen. Myös muiden sovellusten käyttöoikeuksien tarkistaminen ja sovellusten toimintojen tarkastelu olisi yksi iso osa-alue, jonne kehitystyötä voisi suunnata. Se vaatisi myös järjestelmätason oikeuksia ja olisi jo niin iso kokonaisuus, että se kannattaisi eriyttää omaksi projektikseen.

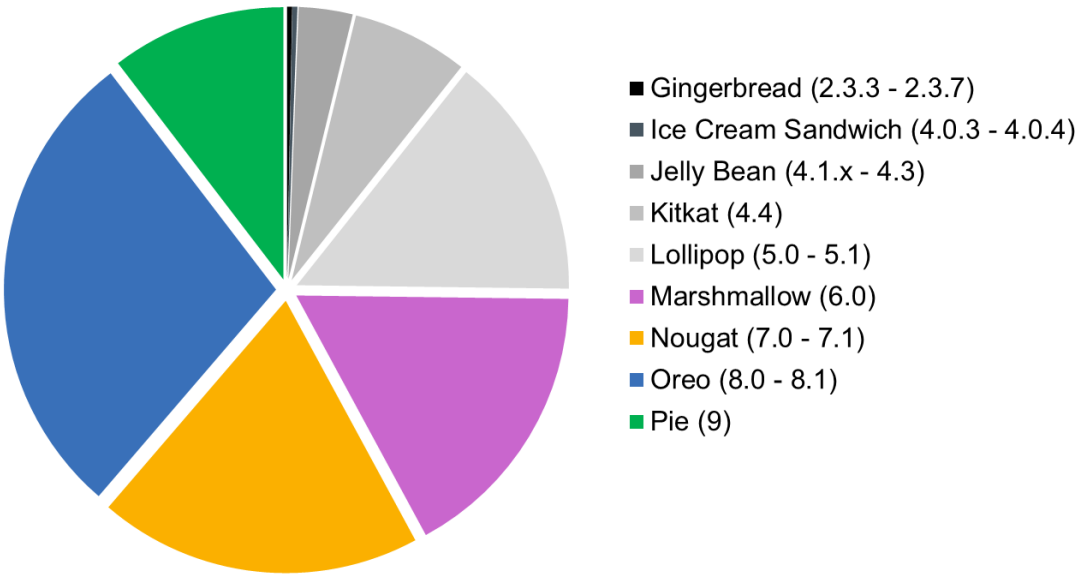
Järjestelmätasolla tehtävät tarkistukset voivat olla hankalampia implementoida valmiiden rajapintojen puuttuessa ja mahdolliset päivitykset, erityisesti Android-versiosta uudempaan, voivat rikkoa nämä tarkistusmenetelmät. Tarkistusmenetelmät voivat olla myös laitekohtaisia, jolloin sama tarkistusmenetelmä ei toimi välttämättä kahden eri valmistajan laitteen välillä, vaikka molemmissa olisi esimerkiksi Android 9 -versio käytössä.

4. SOVELLUKSEN KUVAUS JA IMPLEMENTAATIO

Projekti, jolle sovellus ja dokumentaatio tehtiin tässä diplomityössä, tarvitsi kirjalliset osuudet ja sovelluksen englanninkielisenä. Tästä syystä sovelluksesta esitellyt kaaviot ja käyttöliittymäelementtien tekstiosat ovat englanniksi. Sovelluksen työnimi on myös englanninkielinen, RiskLevelDisplayer.

4.1. Sovelluksen vaatimukset ja rajoitukset

Tässä työssä kehitetyn sovelluksen tarkoituksena oli selvittää Android-laitteen tietoturvan riskitaso, esittää se käyttäjälle sekä lähettää se tarvittaessa eteenpäin toiselle Android-sovellukselle. Kehitetty sovellus tulisi pystyä asentamaan kaikille Android-laitteille, mutta työn rajaamiseksi tuettaviksi Android-versioiksi valittiin versiot 6–9. Nämä Android-versiot kantavat koodinimiä Marshmallow, Nougat, Oreo ja Pie. Sovelluksen tekohetkellä nämä versiot kattoivat noin 75 % käytössä olevista Android-laitteista, kuten kuvassa 20 näkyy. Tiedot kuvaajaa varten haettiin Androidin kehittäjäsivuille kootuista tiedoista [94]. Tiedot on sivun mukaan kerätty seitsemän päivän jaksolta ja tarkastelujakso päättyi 7.5.2019.



Kuva 20. Android-käyttöjärjestelmäversioiden jakaantuminen.

Toinen sovellus, jolle tieto riskitasosta lähetetään, voi tehdä laskeneen tai kohonneen riskitason perusteella laitteen tietoturvaan ja verkkoliikenteeseen liittyviä kovennuksia tai höllennyksiä. Tätä toimintaa toteuttava demo tehtiinkin erääseen toiseen sovellukseen, mutta kehitystyö tätä varten rajattiin kuitenkin lopputyön ulkopuolelle. Toiselle sovellukselle lähetettävän riskitason esittämistapaan otetaan sen sijaan kantaa jatkokehitysideoinnin alla.

Riskitason mittaaminen ja lähettäminen toiseen sovellukseen tuli onnistua siis myös taustatoimintona, eli silloin kun riskitasonmittaussovellus ei ole käyttäjälle näkyvillä aktiviteettimuodossa (engl. *activity*). Tätä varten tarvittiin sovellukseen pääaktiviteetin lisäksi siis palvelu, joka voisi toimia taustalla (engl. *background service*). Jotta palvelu saisi pysyä toiminnassa taustalla Androidin 8.0 -versiossa

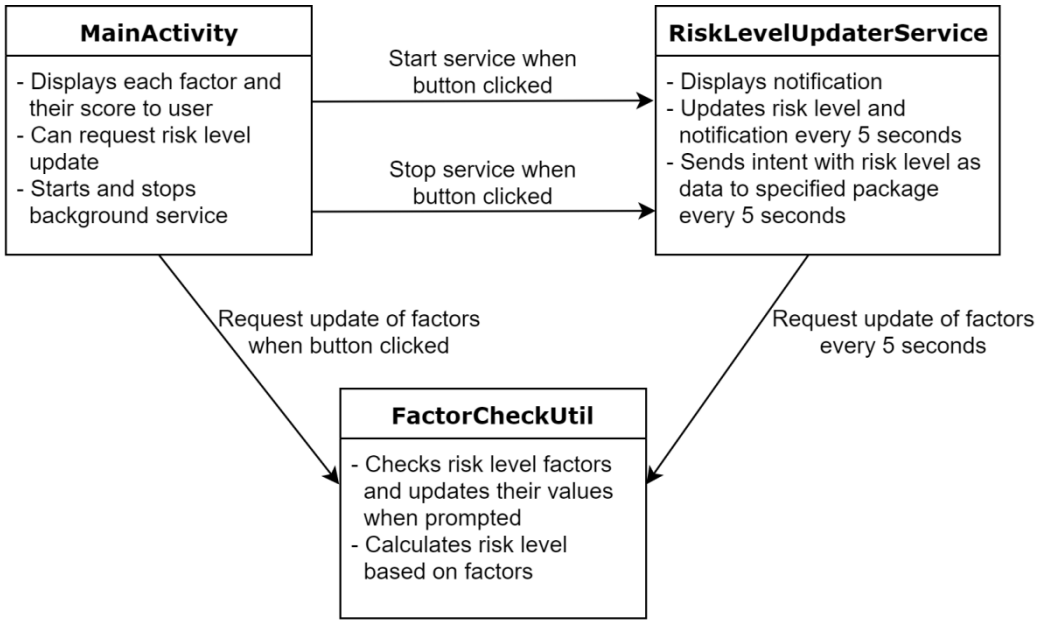
esiteltyjen taustapalvelujen rajoitusten kanssa, sille tuli tehdä myös käyttäjälle näkyvä ilmoitusosa (engl. *notification*) ilmoituspalkkiin [95].

Tässä työssä kehitetyn sovelluksen tuli toimia niin sanotussa geneerisessä Android-laitteessa, eli sen tulee olla ladattavissa ja asennettavissa jonkin sovelluskaupan kautta mille tahansa Android-laitteelle kunhan se täyttää käyttöjärjestelmävaatimukset (Android 6-9). Kääntäen tämä tarkoittaa sitä, että sovellus ei saanut olla vain tietylle Android-laitteelle rakennettu järjestelmätason sovellus, joka vaatisi root-oikeudet tai sovelluksen sisällyttämistä Android-laitteen järjestelmäkuvaan, joka sitten asennettaisiin laitteelle ylikirjoittamalla vanha järjestelmäkuva (engl. *flashing*). Tämän rajoituksen takia tietoturvan riskitason mittaamiseen tuli käyttää Androidin sovellusten ohjelmointirajapintaa, sillä se on tavallisten, ei-järjestelmätason sovellusten käytettävissä.

Edellä mainittuja vaatimuksia lukuun ottamatta sovelluksen suuntaa ja toteuttamistapaa ei rajattu. Sovelluksen tarvittavat toiminnot ja tarkempi toimintamalli tarkentuivatkin sitä mukaa kun sovellusta kehitettiin. Esimerkiksi tietoturvan osatekijöitä, niiden esittämistapaa, tiedonvälitystä toiseen sovellukseen ja taustapalvelun toimintaa ei ollut tarkemmin määritelty.

4.2. Kuvaus

Sovelluksen toiminta voidaan jakaa kolmen luokan (engl. *class*) toimintoihin. Näitä luokkia ovat pääaktiviteetti (MainActivity), taustapalvelu (RiskLevelUpdaterService) ja riskitason tarkistamiseen ja laskemiseen tehty luokka (FactorCheckUtil). Näiden luokkien toimintaa ja vuorovaikutusta keskenään esitellään kuvassa 21.



Kuva 21. Sovelluksen rakenne ja eri luokkien toiminnot.

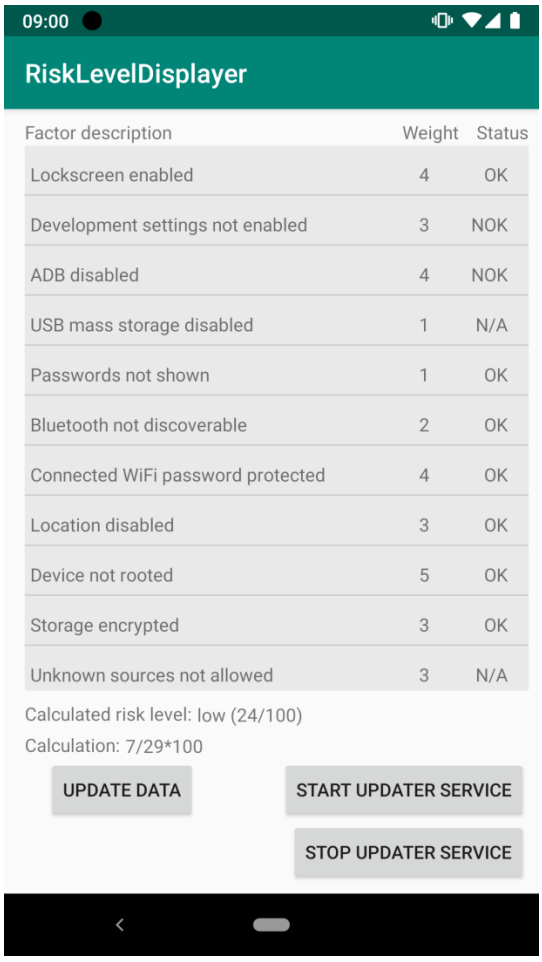
Pääaktiviteetti on näkymä, joka avautuu kun sovellus käynnistetään. Sitä esitellään kuvassa 22. Pääaktiviteetissa näytetään listamuodossa jokainen tietoturvariskin osatekijän kuvaus (engl. *factor description*), sen painotusarvo (engl. *weight*)

riskitason laskussa ja osatekijän tila (engl. *status*). Nämä 11 osatekijää eivät ole missään erityisessä järjestyksessä. Painoarvo on luku väliltä 1–5, ja mahdolliset tilat, mitä osatekijä voi saada, on esitelty taulukossa 3.

Taulukko 3. Tietoturvan osatekijöiden tilat ja niiden selitykset.

Tila	Selitys
OK	Osatekijä on kunnossa
NOK (not OK)	Osatekijä on epäkunnossa
N/A (not available)	Osatekijän tilaa ei voida hakea järjestelmän rajoitusten tai käyttöjärjestelmäversion yhteensopimattomuuden takia

Pääaktiviteetissa näytetään laskettu riskitaso (engl. *calculated risk level*), joka on lukuarvo väliltä 0–100. Lasketun riskitason alapuolella näytetään luvut, joiden perusteella riskitaso on laskettu. Pääaktiviteetissa on myös käytössä kolme painiketta, joilla voi päivittää riskitason osatekijöiden tilat, käynnistää taustapalvelun ja sammuttaa taustapalvelun. Riskitason osatekijöiden tilat tarkistaa ja riskitason laskemisen hoitaa FactorCheckUtil-luokka, jota pääaktiviteetti ja taustapalvelu tarvittaessa kutsuvat.



Kuva 22. Sovelluksen pääaktiviteetti.

Riskitason osatekijöillä on kovakoodatut painoarvot, jotka vaikuttavat lopulliseen riskitason laskemiseen. Nämä painoarvot ovat näkyvissä pääaktiviteetissa. Painoarvo on mietitty jokaisen osatekijän kohdalla sen tuoman tietoturvariskin todennäköisyyden ja vaikutuksen tulona, eli kuten laskukaavassa 1 alla.

$$Riskin\ painoarvo = todennäköisyys * vaikutus \quad (1)$$

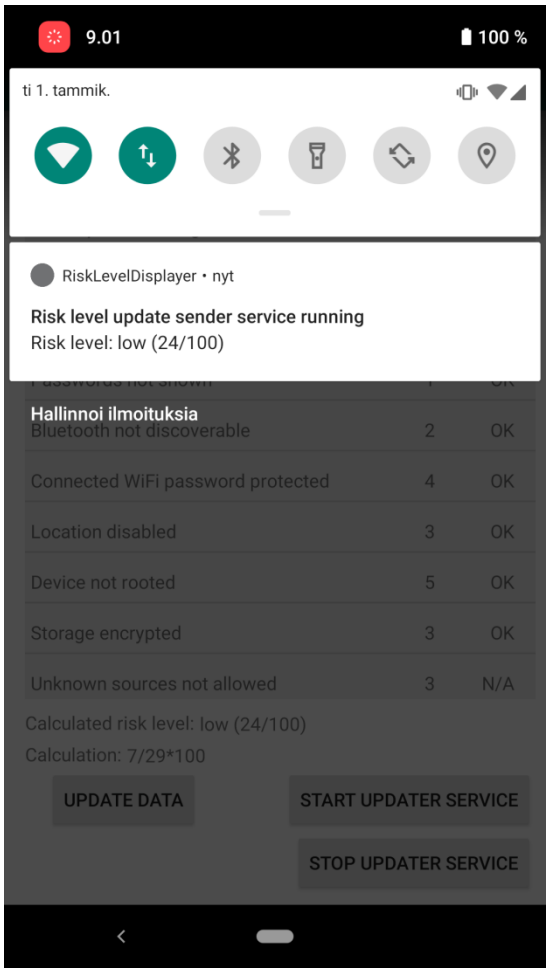
Tämä perustuu yleiseen riskinmääritysmalliin, jota esimerkiksi OWASP-säätiö käyttää myös riskinmääritysmallinsa pohjana [14]. Työn pääpaino ei ollut riskitason laskemisessa ja sen esitysmuodossa vaan saatavilla olevien osatekijöiden selvittämisessä, joten tässä esitelty malli on enemmän luonnostyylinen, jota on tarkoitus kehittää informatiivisemmaksi ja helppotajuisemmaksi. Tähän viitataan myöhemmin myös jatkokehitysosiossa.

Laitteen tietoturvan riskitaso ilmaisee, kuinka suuri riski laitteessa on tietoturvan osatekijöiden kannalta joutua tietoturvahyökkäyksen kohteeksi. Kääntäen tämän voi esittää myös niin, että riskitaso kertoo, kuinka helppo mahdollisen hyökkääjän on päästä laitteen tietoihin käsiksi ja miten suurta vahinkoa laitteelle voi tätä kautta tehdä. Riskitaso, joka käyttäjälle esitetään, lasketaan summaamalla epäkunnossa olevien osatekijöiden painoarvo ja jakamalla se kaikkien saatavilla olevien osatekijöiden painoarvojen summalla. Saatu luku skaalataan välille 0–100 kertomalla se sadalla ja luvun rinnalla esitetään myös sanallinen kuvaus riskitasosta seuraavasti: matala (pistearvo 0–24, engl. *low*), keskitasoinen (pistearvo 25–49, engl. *medium*), korkea (pistearvo (50–74, engl. *high*) ja erittäin korkea (pistearvo 75–100, engl. *very high*). Osatekijät, joiden tietoja ei ole saatavilla käyttöjärjestelmäversiosta tai muusta syystä johtuen, jätetään riskitason laskemisen ulkopuolelle. Laskukaavan voi esittää kaavan 2 mukaisesti, jossa NOK tarkoittaa tilaltaan epäkunnossa olevien riskitekijöiden painoarvojen summaa ja OK tarkoittaa kunnossa olevien riskitekijöiden painoarvojen summaa.

$$Riskitaso = \frac{\mathrm{NOK}}{\mathrm{NOK+OK}} * 100 \quad (2)$$

Taustapalvelu eli RiskLevelUpdaterService käynnistetään pääaktiviteetin kautta. Päällä ollessaan taustapalvelu näkyy ilmoituspalkissa ilmoituksena. Tätä esitellään kuvassa 23. Palvelu pyytää riskitason osatekijöiden päivitystä viiden sekunnin välein ja päivittää saadun tiedon ilmoitukseen. Samalla se myös lähettää riskitasoluvun (0–100) eteenpäin intent-muodossa [96] toiselle sovellukselle, jonka nimi on kovakoodattu palveluun. Tämä toinen sovellus voi sitten tehdä riskitason muutoksien perusteella muutoksia laitteen toimintaan ehkäisten mahdollisia tietoturvauhkia. Taustapalvelu pysyy päällä vaikka pääaktiviteetista poistuttaisiin kotinäyttöön tai johonkin toiseen sovellukseen. Palvelun sulkeminen tapahtuu pääaktiviteetin kautta.

Tämän työn yhteydessä toteutettiin demo, jossa toinen sovellus, jolle riskitasoluvun sisältävä tieto lähetettiin, oli järjestelmätason sovellus. Se pystyi kohonneen riskitason perusteella vaihtamaan Android-laitteen VPN-asetuksia. Esimerkiksi riskitason ollessa matala, VPN ei ollut pakollinen. Korkeammalla riskitasolla VPN pakotettiin päälle ja korkeimmalla riskitasolla pakotettiin päälle karanteenitason VPN-profiili, joka esti pääsyn kriittisiin sisäverkon tietoihin tietomurtojen estämiseksi.



Kuva 23. Taustapalvelu toiminnassa.

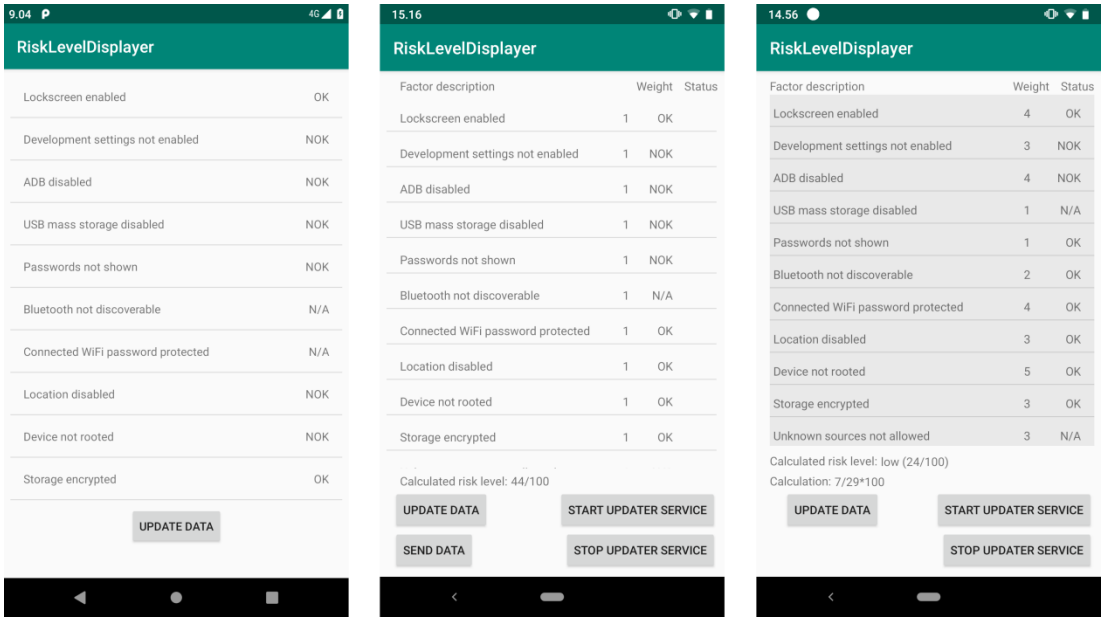
Sovellus asennetaan ja käynnistetään kuten mikä tahansa muu Android-sovellus, jolloin se aukaisee pääaktiviteetin, joka on esitelty kuvassa 22. Kehitysvaiheessa sovellus asennettiin Android-laitteelle tietokoneelta Android Studio -ohjelmointiympäristön kautta.

4.3. Implementaatio

Sovellus tehtiin Android Studio -ohjelmointiympäristöllä Java-ohjelmointikieltä käyttäen Windows- ja Linux-tietokoneilla. Sovelluksen testaamiseen käytettiin Bittium Tough Mobile -, Bittium Tough Mobile 2 - ja OnePlus 6 -älypuhelimia, joissa oli Android-käyttöjärjestelmäversioita Android 6:sta 9:ään. Versionhallintapalveluna toimi Gitlab, joka ei tosin ollut käytössä aivan projektin alusta asti.

Suurin osa sovelluksen tekemiseen käytetystä ajasta meni selvitystyöhön siitä, miten eri tietoturvan osatekijöitä pystytään mittaamaan sovellusten ohjelmointirajapintaa käyttäen. Suurinta osaa tarkistettavaksi halutuista osatekijöistä ei pystytty lopen selvittämään tällä tavalla Androidin sovellusten hiekkalaatikoinnin vuoksi. Tavallisilla Android-sovelluksilla on hyvin rajatut oikeudet tietoturvaa koskeviin tietoihin, ja joitakin tietoturvan osatekijöitä varten joutui käyttämään kiertoteitä halutun tiedon selvittämiseen.

Sovelluksen kehittymisen eri vaiheita näkyy kuvassa 24. Vasemmanpuoleisessa ruutukaappauksessa näkyy jo osatekijät, jotka päätyivät lopulliseen versioon, mutta joidenkin osatekijöiden tarkistuksessa oli vielä puutteita erikoistilanteissa. Tästä esimerkkinä Bluetoothin näkyvyys, joka on N/A-tilassa. Painoarvot, taustapalvelu ja riskitason laskeminen olivat vielä toteuttamatta. Keskimmäisessä ruutukaappauksessa painojen toteutus oli vielä kesken, taustapalvelu toimi vain sovelluksen pääaktiviteetin ollessa päällä ja intentin lähetys toiselle sovellukselle toimi pelkästään ”Send data”-painiketta painamalla, eli lähetystä ei ollut integroitu vielä taustapalveluun kuten oli tarkoitus. Joidenkin osatekijöiden tarkistuksessa oli myös vielä epävakautta. Oikeanpuolimmaisessa ruutukaappauksessa sovellus on jo stabiili toiminnaltaan, riskitaso lasketaan painoarvojen mukaan, käyttöliittymää on tehty hieman selkeämmäksi ja osatekijälista skaalautuu pääaktiviteetissa oikein erikokoisille näytöille, myös vaakasuunnassa käytettäessä. Taustapalvelun ja pääaktiviteetin riskitason hakua FactorCheckUtil-luokalta piti vielä parantaa tästä versiosta, mutta se ei enää näkynyt käyttöliittymän puolella.



Kuva 24. Eri versioita sovelluksesta, vanhin vasemmalla ja uusin oikealla.

5. SOVELLUKSEN ARVIOINTI JA JATKOKEHITYS

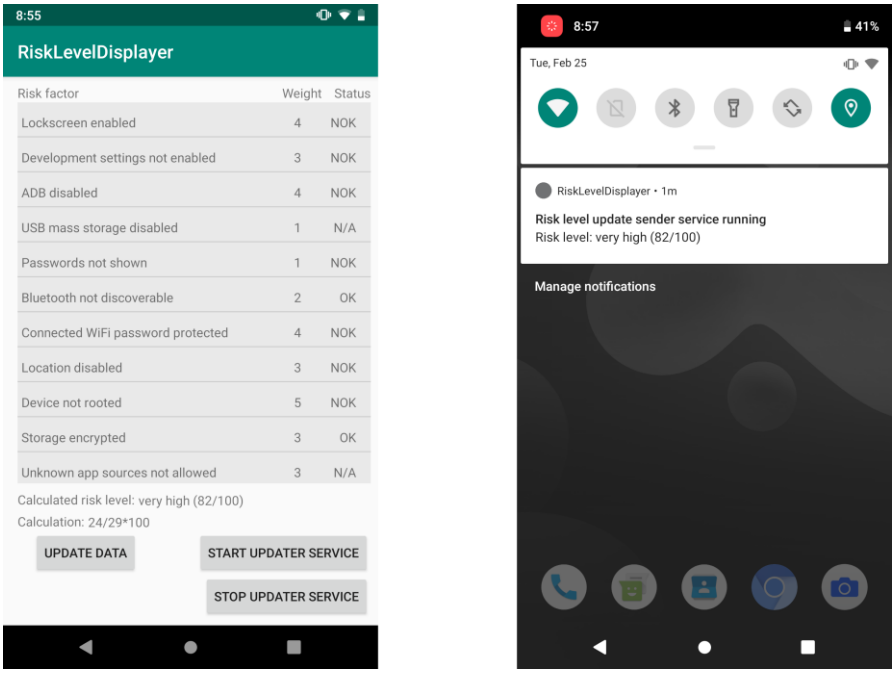
Työn tuloksena oli Android-sovellus tietoturvan riskitason mittaamiseen, mutta sen laajempi merkitys oli selvittää pohja tämän tyylisen sovelluksen pidemmälle menevälle kehitystyölle. Tarkoitus oli siis selvittää, oliko sovellusideassa potentiaalia ja mihin suuntaan sitä kannattaisi jatkossa rakentaa.

5.1. Sovelluksen toiminnan arviointi

Työn teknisenä tuloksena oli Android-sovellus, joka mittaa laitteen tietoturvan riskitasoa ja jonka voi asentaa mihin tahansa Android 6–9 käyttöjärjestelmäversiolla olevaan laitteeseen. Sovellus mittaa laitteesta 11 aiemmin esiteltyä tietoturvaan liittyvää osatekijää ja näyttää listauksen niiden tilasta sekä laskee näiden perusteella laitteelle sen riskitason. Sovellusta voi käyttää myös taustalla, jolloin se näkyy ilmoituspalkissa ilmoittaen riskitason. Riskitaso päivitetään ilmoituspalkkiin viiden sekunnin välein. Sovellus voi samalla myös lähettää lasketun riskitasoluvun toiselle sovellukselle intent-muodossa ja tämä toinen sovellus voi esimerkiksi tehdä riskitason perusteella kovennuksia ja höllennyksiä laitteen toimintaan tietoturvamielessä, jos se on järjestelmätason sovellus. Tämän työn sovellukselle asetetut vaatimukset siis täyttyivät.

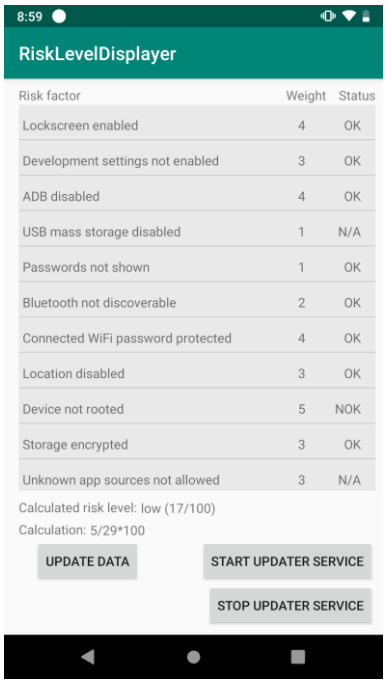
Sovelluksen toimivuutta arvioidaan ja esitellään seuraavaksi neljän esimerkin avulla, joissa käydään läpi kaikki osatekijät sekä taustapalvelun toiminta. Ensimmäisessä esimerkissä on käytössä Bittium Tough Mobile 2 -älypuhelin, jonka riskitaso on alkutilanteessa nostettu tahallisesti mahdollisimman korkealle, ja riskitasoa lasketaan sitten muuttamalla osatekijöiden tiloja. Toisessa esimerkissä esitellään sovelluksen tilaa Bittium Tough Mobile -älypuhelimella, kun riskitaso on matala ja tuntemattomat sovelluslähteet kytketään päälle. Kolmannessa esimerkissä esitellään Bluetoothin löydettävyyttä OnePlus 6 -älypuhelimella. Neljännessä esimerkissä esitellään riskitasotiedon lähetystä taustapalvelun kautta ja sen vastaanottoa toisessa sovelluksessa.

Ensimmäisen esimerkin alkutilannetta on esitelty kuvassa 25, jossa vasemmalla näkyy osatekijöiden tilat ja laskettu riskitaso (82). Sanallinen kuvaus riskitasolle on siis ”erittäin korkea” (*very high*). Kuvassa oikealla näkyy taustapalvelu toiminnassa, kun sovelluksen pääaktiviteetista on poistuttu kotinäyttöön. Taustapalvelu näyttää samaa riskitasoa (82). Tässä alkutilanteessa puhelimen riskitasoa nostavat seuraavat osatekijät; näytön lukitus ei ole käytössä, kehittäjäasetukset ovat käytössä, ADB on käytössä, salasanojen näkyvyys on käytössä, puhelin on yhteydessä avoimeen wifi-verkkoon, sijainti on käytössä ja puhelin on rootattu. USB-massamuistin käytön tilaa ei pystytty mittaamaan, koska puhelin ei ollut kytkettynä USB-yhteydellä tietokoneeseen, joten sen tila näkyy N/A-muodossa. Tämän osatekijän mittaamisessa oli epävakautta, eikä sen toimivuutta päästy esittelemään näissä esimerkeissä. Tuntemattomien sovelluslähteiden tilaa ei myöskään voida tässä puhelimessa mitata, sillä siinä on käytössä Android 9 -versio. Laitteen tallennustila on salattu ja salausta ei voi kytkeä pois päältä. Puhelimen Bluetooth ei ole löydettävässä tilassa, sillä tässä puhelinmallissa se on löydettävissä vain, kun Bluetooth-valikko on auki ja Bluetooth on kytketty päälle. Tämän osatekijän mittausta ei voi siten esitellä tällä puhelinmallilla.



Kuva 25. Ensimmäisen esimerkin alkutilanne, erittäin korkea riskitaso.

Ensimmäisessä esimerkissä puhelimen riskitasoa laskettiin alkutilanteen jälkeen muuttamalla kaikkien muutettavissa olevien osatekijöiden tilaa parempaan suuntaan. Näytön lukitus otettiin käyttöön, kehittäjäasetukset, ADB ja salasanojen näkyvyys otettiin pois käytöstä, wifi-verkko vaihdettiin avoimesta verkosta salattuun ja sijainti otettiin pois käytöstä. Puhelimen roottausta ei voitu kuitenkaan poistaa tässä esimerkissä. Edellä mainitut muutokset laskivat laitteen riskitasoa. Lopputilannetta on esitelty kuvassa 26, jossa näkyvät osatekijöiden muuttuneet tilat ja riskitason laskenut lukuarvo (17). Sanallinen kuvaus riskitasolle on ”matala” (*low*).



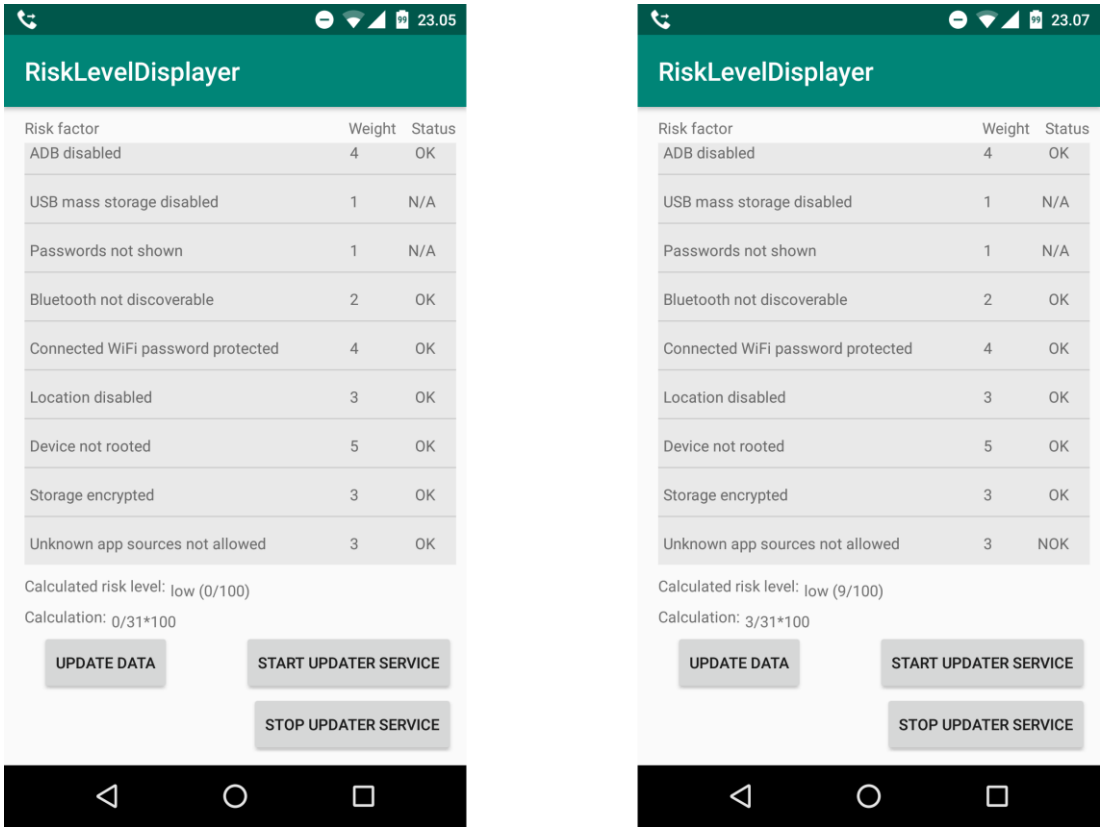
Kuva 26. Ensimmäisen esimerkin lopputilanne, matala riskitaso.

Toisessa esimerkissä esitellään tuntemattomien sovelluslähteiden tarkistusta Bittium Tough Mobile -älypuhelimella, jossa on käyttöjärjestelmänä Androidin 6.0.1-versio. Android 6–7.1.1 -versioissa tuntemattomien sovelluslähteiden salliminen on mahdollista tarkistaa sovellusten ohjelmointirajapinnan kautta. Sitä uudemmissa se ei ole mahdollista, sillä asetus muuttui sovelluskohtaiseksi. Tästä kerrottiin myös luvussa 3.11.

Kuvassa 27 vasemmalla näkyy riskitason tilanne, kun tuntemattomat sovelluslähteet eivät ole sallittuja. Riskitaso on tässä vaiheessa matalin mahdollinen (0). Kuvassa oikealla on riskitasonäkymä sen jälkeen kun tuntemattomat sovelluslähteet on käyty sallimassa puhelimen asetuksista. Riskitaso on kohonnut (9), mutta se luokitellaan edelleen sanallisesti matalaksi.

Tässä esimerkissä salasanojen näkyvyyden tila ei ole saatavilla (N/A). Tämä johtuu Androidin rajoituksesta tuon osatekijän tarkistuksen suhteen. Jos salasanan näkyvyyden tilaa ei ole muutettu kertaakaan laitteen käyttöönoton aikana, tämän osatekijän tarkistukseen käytettävä muuttuja Androidin ohjelmointirajapinnassa ei ole saanut mitään arvoa, vaikka salasanojen näkyvyys on todellisuudessa joko päällä tai pois päältä.

Kuvakaappauksista voi myös nähdä miten sovellus skaalautuu pienemmälle näytölle. Osatekijälistan kaikki 11 osatekijää eivät mahdu kokonaisuudessaan näkyville, vaan lista on pitänyt vierittää sen loppuun, jotta viimeinen osatekijä eli tuntemattomat sovelluslähteet näkyy. Kaikki näytön tekstielementit eivät asetu vielä täysin oikeille kohdille erikokoisilla näytöillä, vaan riskitason sanallinen ja numeerinen kuvaus sekä riskitason laskemisen näyttävä laskukaava ovat hieman alempana kuin niiden pitäisi olla.

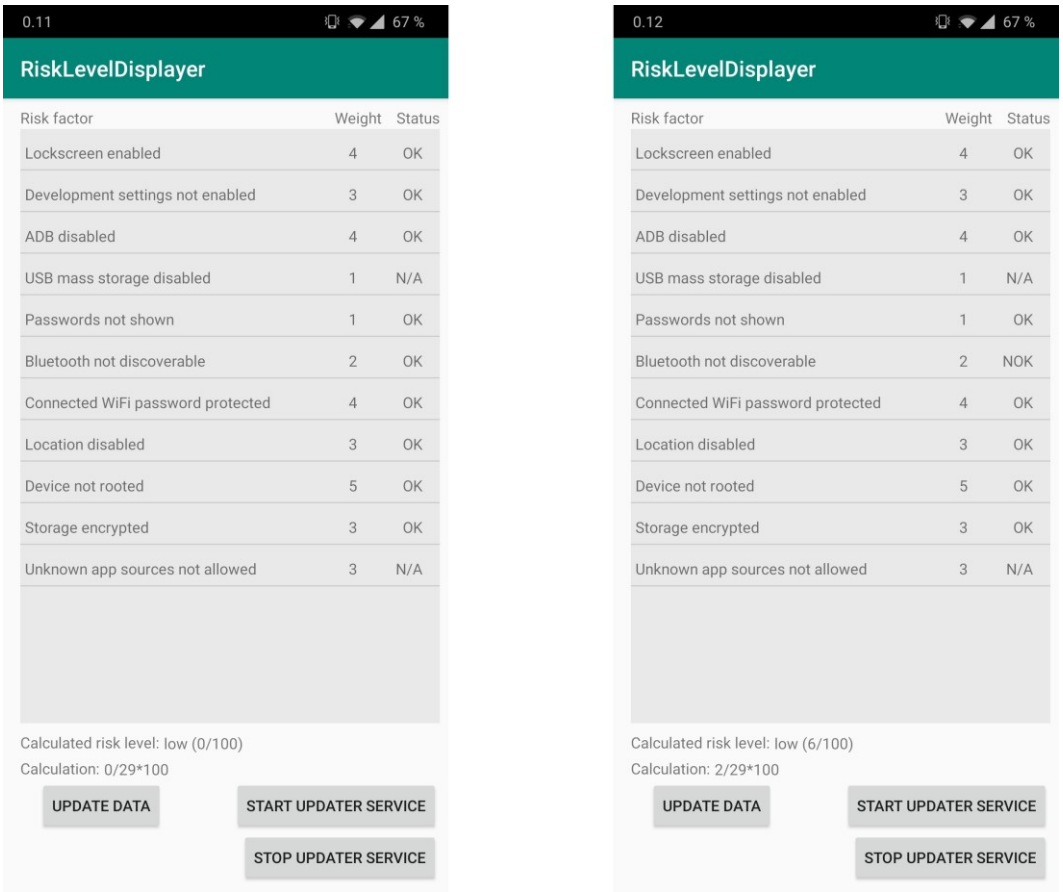


Kuva 27. Matalin mitattava riskitaso ja tuntemattomien sovelluslähteiden käyttö.

Kolmannessa esimerkissä esitellään Bluetoothin löydettävyyttä OnePlus 6 -älypuhelimella. Tällä puhelimella Bluetooth voidaan kytkeä löydettäväksi erillisellä valinnalla asetuksissa niin, että Bluetooth-valikon ei tarvitse olla jatkuvasti näkyvillä, toisin kuin Bittium Tough Mobile - ja Bittium Tough Mobile 2 -puhelimilla.

Kuvan 28 alkutilanteessa vasemmalla Bluetooth ei ole löydettävässä tilassa. Kuvan oikeanpuoleisessa kuvakaappauksessa Bluetooth taas on laitettu löydettävään tilaan, jolloin se nostaa riskitason nollasta kuuteen. Tässä puhelimessa näytön koko on myös hieman suurempi kuin edellisen esimerkin malleissa, ja lisäksi tekstin ja näytön kohteiden kokoa on muutettu hieman pienemmiksi puhelimen asetuksista. Osatekijälista mahtuu näin ollen näytölle kokonaan, mutta listaelementille varattu koko on niin iso, että sen alaosaan jää tyhjää tilaa. Myös riskitasosta kertovat kuvaukset ja laskukaava ovat hieman liian alhaalla.

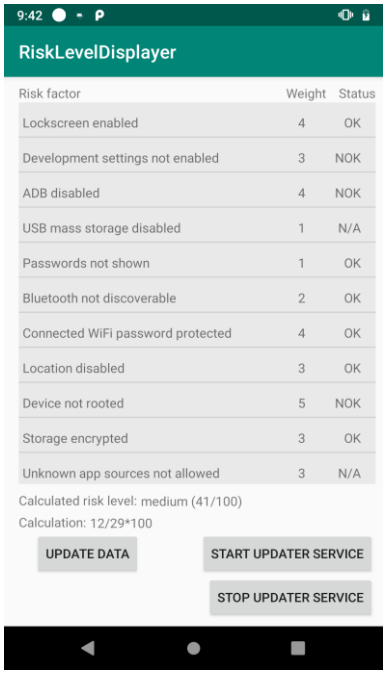
Tässä esimerkissä puhelin on roottaamaton kuten edellisessäkin. Ensimmäisen esimerkin puhelin oli rootattu ja tämän eron voi nähdä kyseisen osatekijän tilassa kuvien 26, 27 ja 28 välillä.



Kuva 28. Bluetoothin löydettävyyden kytkeminen päälle.

Neljännessä esimerkissä esitellään tarkemmin riskitasotiedon lähetystä intent-muodossa ja sen vastaanottamista toisessa sovelluksessa. Vastaanottamisen esittelyä varten tehtiin toinen sovellus, joka kykenee vastaanottamaan riskitasotiedon ja lokittaa sitten saadun riskitiedon niin, että se voidaan lukea *logcat*-työkalulla. Kuvassa 29 on kuvakaappaus riskitason mittaussovelluksesta Bittium Tough Mobile 2 -puhelimella. Riskitasoksi on laskettu 41 ja sovelluksen taustapalvelu

(RiskLevelUpdaterService) sekä riskitasotiedon vastaanottavan sovelluksen taustapalvelu (ReceiverService) on käynnistetty. Ne näkyvät ikoneina ilmoituspalkin vasemmassa reunassa.



Kuva 29. Mitattu riskitaso, joka lähetetään taustapalvelussa eteenpäin.

Kuvassa 30 on kuvakaappaus *logcat*-työkalun tuottamasta lokista, kun riskitason mittaussovelluksen taustapalvelu on käynnistetty ja se on ollut toiminnassa hieman yli kymmenen sekuntia. Taustapalvelun ollessa toiminnassa se pyytää riskitasotiedon päivitystä FactorCheckUtil-luokalta 5 sekunnin välein. FactorCheckUtil tarkistaa osatekijöiden tilat ja palauttaa riskitasoluvun taustapalvelulle. Saatuaan riskitasoluvun taustapalvelu lähettää sen eteenpäin toiselle sovellukselle intent-muodossa. Nämä tapahtumat lokitetaan niin, että ne ovat luettavissa *logcat*-työkalulla. Kun toinen sovellus saa riskitasotiedon oman taustapalvelunsa (ReceiverService) kautta, se lokittaa myös tämän tiedon samalla tavalla. Näitä lokitapahtumia esitellään kuvassa 30, jossa *logcat*-työkalun tulosteesta on seulottu *grep*-komennolla riskitason mittaussovelluksen taustapalvelun (RiskLevelUpdaterService) ja vastaanottavan sovelluksen taustapalvelun (ReceiverService) lokitiedot. Tulosteesta on poistettu vielä erään järjestelmäsovelluksen näistä palveluista tekemät ylimääräiset lokimerkinnät *grep -v* -komennolla.

```
[redacted]:~$ adb logcat | grep 'RiskLevelUpdaterService\|ReceiverService' | grep -v servicetracker
02-25 21:40:35.349 17085 17085 D RiskLevelUpdaterService: Service started
02-25 21:40:35.372 17085 17695 D RiskLevelUpdaterService: Risk level updated to: 41
02-25 21:40:35.400 17085 17695 D RiskLevelUpdaterService: Sent intent with data: 41
02-25 21:40:35.418 17592 17696 D ReceiverService: Received intent. Intent data: 41
02-25 21:40:40.409 17085 17695 D RiskLevelUpdaterService: 5 seconds passed
02-25 21:40:40.437 17085 17695 D RiskLevelUpdaterService: Risk level updated to: 41
02-25 21:40:40.478 17085 17695 D RiskLevelUpdaterService: Sent intent with data: 41
02-25 21:40:40.495 17592 17699 D ReceiverService: Received intent. Intent data: 41
02-25 21:40:45.485 17085 17695 D RiskLevelUpdaterService: 5 seconds passed
02-25 21:40:45.515 17085 17695 D RiskLevelUpdaterService: Risk level updated to: 41
02-25 21:40:45.554 17085 17695 D RiskLevelUpdaterService: Sent intent with data: 41
02-25 21:40:45.584 17592 17702 D ReceiverService: Received intent. Intent data: 41
```

Kuva 30. Kuvakaappaus riskitasotiedon lähetyksestä ja vastaanotosta *logcat*-lokissa.

Alun kolmessa esimerkissä näytettiin yhdeksän osatekijän tilan tarkistuksen toiminta. Osatekijät, joiden tarkistuksen toimivuutta ei voitu niissä esitellä, ovat USB-massamuistin käyttö ja tallennustilan salauksen käyttö. USB-massamuistin käytön tarkistuksessa oli epävakautta, jota ei saatu tämän työn aikarajoitteissa korjattua. Epävakaus johtuu joko Androidin rajoituksesta, kuten todettiin salasanojen näkymisen osalta, tai osatekijän tarkistusmetodin viallisesta toteutuksesta. Tallennustilan salaus oli oletuksena käytössä kaikissa puhelimissa, joilla sovellusta päästiin testaamaan, eikä sitä ollut mahdollista kytkeä mistään niistä pois päältä. Näin ollen tämän osatekijän tarkistuksen toimivuuttaa ei päästy testaamaan luotettavasti. Nämä kaksi osatekijää pois lukien osatekijöiden tarkastukset toimivat niin kuin niiden piti toimia.

Riskitasotiedon välittäminen toiseen sovellukseen toimi myös vaatimusten mukaisesti. Riskitasotiedon välittävän palvelun elossapysymisen kanssa oli aluksi ongelmia, sillä käyttöjärjestelmä lopetti aina sen toiminnan hetken kuluttua aloituksesta. Tämä vaati siten ilmoitusosan toteuttamista taustapalvelulle. Tämän ansiosta taustapalvelu pysyy päällä niin kauan, että käyttäjä käy sen erikseen lopettamassa.

5.2. Sovelluksen esittämän riskitason arviointi

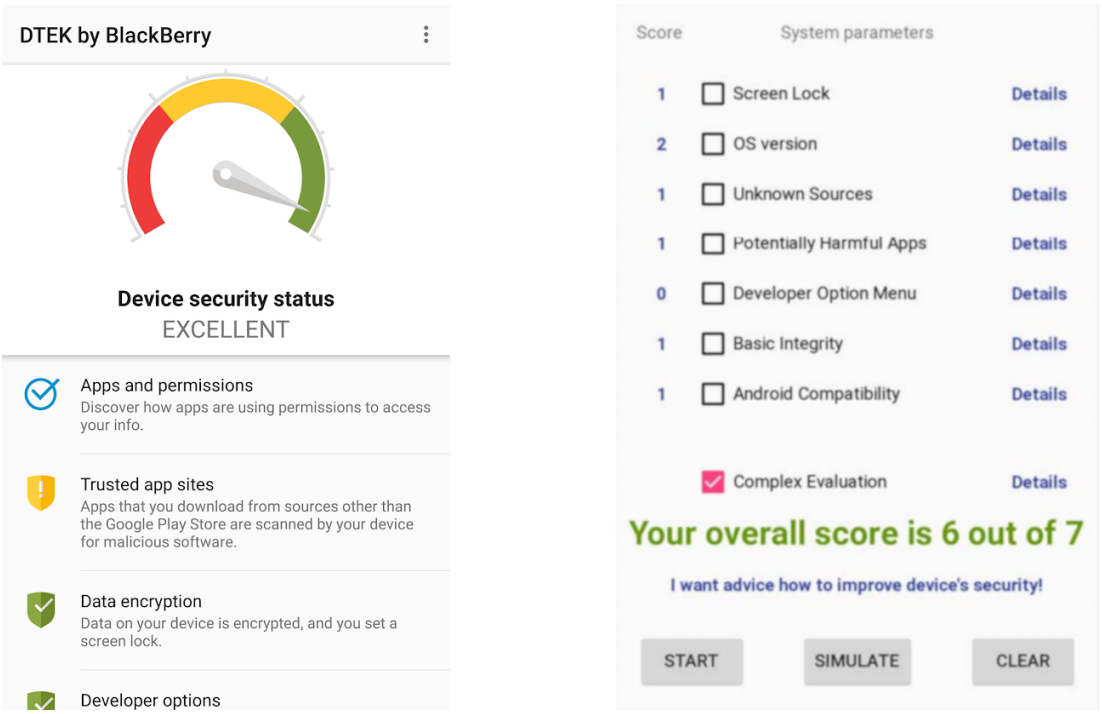
Riskitason mittaussovellus antaa laitteen tietoturvan riskitasoksi luvun väliltä 0–100, nollan ollessa matalin ja sata korkein mahdollinen riskitaso. Lukuarvo perustuu 11 tietoturvan osatekijän tilaan, jotka mitataan laitteesta. Todellisuudessa huomioon otettavia osatekijöitä olisi paljon enemmän kuin nämä 11. Androidin ohjelmointirajapinnalla on mahdollista tarkastella ja mitata useampia osatekijöitä kuin sovellukseen toteutetut 11 osatekijää. Näitä olisivat esimerkiksi Android-version ja tietoturvapäivitysten päivämäärän tarkistaminen. Tavallisella, ei-järjestelmätason sovelluksella tarkistettavista osatekijöistä kerrotaan lisää jatkokehitysosiossa luvussa 5.3.1.

Aiemmin mainitut 11 osatekijää ja luvussa 5.3.1. mainitut osatekijät liittyvät laitteen suojausominaisuuksiin, tietoturvapäivityksiin ja käytössä oleviin yhteyksiin. Jos laitteen riskitasoa haluttaisiin kuvata kokonaisvaltaisemmin, tarkasteluun pitäisi ottaa myös laitteeseen asennetut sovellukset sekä laitteen lokitiedot. Jotkin sovellukset voidaan todeta suoraan tietoturvariskeiksi tarkistamalla niiden luotettavuus jotakin valmista tietokantaa vasten, johon on kerätty tietoja haitallisiksi tunnetuista sovelluksista. Google Play Protect -järjestelmä tekeekin jo tätä [87]. Aiemmin tunnistamattomien sovellusten toiminnan ja laitteen lokitiedostojen tarkastelu tietoturvapoikkeamien varalta (engl. *anomaly detection*) vaatisi poikkeamien tunnistamista. Yksi tapa tehdä tämä olisi kehittää tarkoitukseen sopiva algoritmi koneoppimisen avulla. Vastaavalla tavalla voitaisiin tarkastella myös laitteen resurssien, kuten keskusmuistin ja prosessorin käyttöä sekä laitteen internet-liikennettä. Nämä menevät kuitenkin enemmän tunkeilijan havaitsemisjärjestelmien piiriin (engl. *intrusion detection system*, *IDS*).

Nämä pidemmän aikavälin tarkastelua vaativat dynaamiset tapahtumat eivät ole yhtä yksiselitteisiä tietoturvan riskitason kannalta kuin tietoturvapäivitysten ajantasaisuus, laitteen tietoturvaominaisuuksien ja turvallisten yhteyksien käyttö ja turvattomien yhteyksien välttäminen. Jos tarkastellaan tässä työssä tehdyn sovelluksen esittämän tietoturvan riskitasoa pelkästään tietoturvaominaisuuksien ja

yhteyksien kannalta, se ei tarkastele riskitasoa niin laajalta alueelta kuin olisi mahdollista. Laitteen suojausominaisuuksista voisi tarkistaa nykyisten osatekijöiden lisäksi käynnistyksenlukijan lukituksen tilan ja tehdasasetusten palauttamisen suojauksen käytön (engl. *factory reset protection*, *FRP*). Yhteyksiin liittyen voitaisiin tarkastella myös VPN:n ja NFC:n käyttöä. Lisäksi voitaisiin tarkastella tietoturvapäivitysten ajantasaisuutta Android-version ja tietoturvapäivitysten päivämäärän perusteella. Androidin ohjelmointirajapinnan kautta näiden osatekijöiden tarkasteleminen pitäisi olla mahdollista. Näistä tarkistettavissa olevista osatekijöistä kerrotaan lisää jatkokehityksessä luvussa 5.3.1.

Aiempia vastaavanlaisia riskitasoa mittaavia sovelluksia ovat Khokhlovin ja Reznikin tekemä sovellus [4] sekä BlackBerryn ”DTEK by BlackBerry”-sovellus [5], joka toimii vain BlackBerryn valmistamilla Android-laitteilla. Näiden sovelluksien päänäkymät on esitelty kuvassa 31.



Kuva 31. DTEK[1]- ja Khokhlovin ja Reznikin[2] sovellusten päänäkymät.

DTEK tarkistaa näytön lukituksen käytön, tallennustilan salauksen käytön, kehittäjäasetuksien käytön, tehdasasetusten palauttamisen suojauksen käytön, käyttöjärjestelmän eheyden, etähallinnan käytön (laitteen paikannusta ja tyhjentämistä varten siltä varalta, että se katoaa tai joutuu varastetuksi) ja tuntemattomien sovelluslähteiden käytön [97]. Sillä voi myös hallinnoida sovellusten käyttöoikeuksia. Sovellus antaa näiden osatekijöiden perusteella laitteen tietoturvalle arvosanan huono, kohtalainen tai erinomainen.

Khokhlovin ja Reznikin sovellus tarkistaa laitteesta näytön lukituksen käytön, Android-version, tuntemattomien lähteiden käytön ja kehittäjäasetusten käytön. Näiden lisäksi se tekee Googlen SafetyNet-kirjaston [98] avulla laitteen käyttöjärjestelmälle eheystestin ja yhteensopivuustestin (engl. *Android compatibility*

[1] https://play.google.com/store/apps/details?id=com.blackberry.privacydashboard
[2] https://www.researchgate.net/publication/323864674_Android_system_security_evaluation

test) sekä tarkistaa, onko laitteelle asennettu mahdollisesti haitallisia sovelluksia (engl. *potentially harmful application, PHA*). Eheystestillä sovellus tarkastelee käyttöjärjestelmän eheyttä. Yhteensopivuustestillä se taas pystyy tarkistamaan käyttöjärjestelmän eheyden lisäksi käynnistyslataajan lukituksen tilan, laitteen valmistajan sertifioinnin sekä käyttöjärjestelmän alkuperäisyyden (engl. *custom ROM*). Laite on sertifioitu jos laitteen valmistajan on läpäissyt laitteella Googlen määrittämät Android-yhteensopivuustestit sekä hakenut tämän jälkeen sertifikaatin laitteelle Googlelta. Mahdollisesti haitallisten sovellusten tarkistus SafetyNet-kirjaston kautta tarkoittaa käytännössä sitä, että testaussovellus tarkistaa onko Google Play Protect -järjestelmä käytössä [99, 100]. Khokhlovin ja Reznikin sovellus antaa jokaisesta tarkistuskohdasta yhden pisteen jos se on kunnossa. Poikkeuksena Android-version uusimmasta versiosta saa kaksi pistettä, edellisestä yhden ja sitä vanhemmista nolla pistettä.

Khokhlovin ja Reznikin sovellus, DTEK-sovellus ja tässä työssä tehty RiskLevelDisplayer mittaavat osin samoja asioita. Kaikilla niillä on myös osa-alueita, joita muut sovellukset eivät mittaa. Sovellusten tekemiä tarkistuksia on vertailtu taulukossa 4.

Taulukko 4. Riskitason mittaussovellusten vertailu.

Osatekijä	RiskLevel -Displayer	Khokhlov ja Reznik	DTEK
Näytön lukituksen käyttäminen	X	X	X
Kehittäjäasetuksien käyttäminen	X	X	X
ADB:n käyttäminen	X		
USB-massamuistin käyttäminen	X		
Salasanan näkymisen tarkistus	X		
Bluetoothin löydettävyyden tarkistus	X		
Avoimen wifi-verkon käyttäminen	X		
Sijainnin käyttäminen	X		
Roottauksen tarkistus	X		
Tallennustilan salauksen käyttäminen	X		X
Tuntemattomien sovelluslähteiden käyttö	X	X	X
Käynnistyslataajan lukituksen tila tarkistus		X	
Laitteen sertifioinnin tilan tarkistus		X	
Google Play Protect -järjestelmän käyttö		X	
Android-version tarkistus		X	
Käyttöjärjestelmän eheyden tarkistus		X	X
Laitteen etähallinnan käyttäminen			X
Tehdasasetusten palauttamisen suojauksen käyttäminen			X

Vaikka tässä työssä tehty sovellus ei ole aikarajoitteiden takia niin laaja kuin se voisi olla, se antaa käyttäjälle jo hyvän kuvan laitteen käytössä olevista suojausominaisuuksista ja yhteyksistä sekä niihin perustuvasta riskitasosta. Sovellusta on mahdollista laajentaa toteuttamalla siihen lisää osatekijöiden tarkistusmetodeja. Sovelluksen kehitystä jatkettiinkin projektissa, jolle tämä työ tehtiin. Jotta riskitason mittaamisesta saataisiin vielä kokonaisvaltaisempi kuin pelkästään käytössä olevia yhteyksiä, tietoturvaominaisuuksia ja tietoturvapäivitysten

ajantasaisuutta tarkastelemalla, tarkasteluun pitäisi ottaa myös laitteen dynaaminen toiminta, eli muiden sovellusten toiminta ja lokitietojen tarkastelu. Niiden vaikutusta riskitasoon on kuitenkin hankalampi määrittää kuin esimerkiksi päällä tai pois päältä olevan näytön lukituksen vaikutusta.

Yhtä yksittäistä tapaa määritellä realistinen riskitaso Android-laitteelle ei ole olemassa. Tämä näkyy myös verrattaessa RiskLevelDisplayer-, DTEK- ja Khokhlovin ja Reznikin sovellusten tekemää riskitason arviointia. Eri osatekijöiden painotusta ja vertailua toistensa suhteen ei voi tehdä täysin yksiselitteisesti, joten määritelty riskitaso voidaan katsoa myös mielipidekysymykseksi. Jos taas jokaisen osatekijän esittäisi käyttäjälle yksittäisinä, erillisinä osatekijöinä tekemättä niistä mistään yhteenvetoa, riskitason hahmottaminen voi olla vaikeaa vähemmän edistyneille käyttäjille. Liian yksinkertaistavan ja liian monimutkaisen riskitason esitystavan väliltä voi löytää silti useimmille käyttäjille sopivan tasapainon. Riskitason esitystavan jatkokehitystä pohditaan lisää luvussa 5.3.2.

5.3. Jatkokehitys

Tässä työssä tehty sovellus oli enemmän konseptin testaamiseen tarkoitettu kokeilu kuin loppuun asti määritelty kaupallinen sovellus. Sen avulla selvisi, että tällaiselle sovellukselle on tarpeeksi tarttumapinta-alaa Androidin sovellusten ohjelmointirajapinnassa, jotta siihen perustuen saataisiin tehtyä järkevä ja hyödyllinen sovellus. Työn taustaksi tehty kirjallinen osuus Androidin tietoturvasta ja sen osa-alueista tukee sovellukseen myöhemmin tehtävää osatekijöiden tarkempaa selittämistä ja perustelua. Pelkkä osatekijöiden listaus ilman lisätietoa niistä jättää niiden mittaamissyyn hämäräksi osalle käyttäjistä.

Implementoidut 11 osatekijän tarkistusta olivat ilmeisimpiä tietoturvaan liittyviä asioita, joiden tarkistamismetodit olivat kohtuullisella työmäärällä tehtävissä. Niiden esitystapa ja riskitason laskeminen toteutettiin myös yksinkertaisella tavalla, koska tarkempaa tapaa ei ollut määritelty eikä ollut selvää näkemystä hyvästä tavasta toteuttaa niitä. Näitä toteuttaessa ja sovellusta demotessa sen edistyessä nousikin uusia ideoita ja näkemyksiä siitä, miten ne voitaisiin tehdä paremmin ja mihin suuntaan sovellusta kannattaisi kehittää.

5.3.1. Muut osatekijät

Tavallisten sovellusten ohjelmointirajapinnalla on rajallinen pääsy Android-laitteen tietoihin, mikä rajoitti alun perin suunniteltujen osatekijöiden mittaamista. Sovellusta kehitettäessä tuli vastaan kuitenkin uusia käyttäjän vaikutuspiirissä olevia ja mitattavissa olevia tietoturvan osatekijöitä. Kaikkia näitä ei ehditty toteuttaa sovellukseen tämän työn aikarajoitusten sisällä, mutta sovelluksen kehitystä jatkettiin projektissa, jolle tämä työ tehtiin. Alla on listattuna tällaisia osatekijöitä.

- Android-versio
- Tietoturvapäivitysten päivämäärä
- OEM-lukituksen tila
- Avainvaraston laitteisto- tai ohjelmistopohjaisuus
- Käynnistyslataajan lukituksen tila
- Lohkolaitteiden eheyden tarkistajan tila (engl. *dm-verity mode*)

- Tehdasasetusten palauttamisen suojaus
- Näytön lukituksen kompleksisuus (vaatii Android 10 -version) [101]
- Bluetoothin käyttö
- NFC:n käyttö
- VPN:n käyttö

Näiden lisäksi roottauksen tarkistamista voisi parantaa, sillä sen tarkistusprosessi on mahdollista kiertää erilaisilla menetelmillä [102]. Tarkistusprosessista on tosin hankala saada täydellistä, sillä rootatulla laitteella on todella laajat keinot tarkistuksen hämäämiseen.

Edellä oleva puuttuvien osatekijöiden listaus ei ole kaikenkattava. Tarkemmin suunnittelemalla ja sovellusrajapinnan järjestelmällisellä läpikäynnillä olisi mahdollista saada esiin tietoturvan osatekijöitä, joita ei ole vielä tullut vastaan.

5.3.2. Käyttökokemus

Sovelluksessa esitetyt osatekijät esitetään lyhyesti listamuodossa ilman tarkempia selityksiä. Tämä esitysmuoto ei ole kovin selkeä ja osatekijät ja niiden tilat vaatisivat tarkemman selityksen. Parannus tähän ongelmaan voisi olla erillinen infopainike osatekijöille, jonka kautta saisi lisätietoja ja ohjeet osatekijän tilan korjaamiseen, jos sen tila on epäkunnossa. Lisäksi joidenkin osatekijöiden kohdalla olevat kaksoisnegaatiot tekevät niiden tulkinnasta hankalaa, kuten esimerkiksi Bluetoothin ollessa löydettävissä. Tällöin osatekijän kuvaus (*Bluetooth not discoverable*) tila ei ole kunnossa (NOK). Tämä esitystapa valittiin sen takia, että NOK viittaisi aina siihen, että osatekijä ei ole kunnossa, mutta se heikentää osatekijöiden kuvauksen ymmärrettävyyttä.

Osatekijät olisi myös hyvä jäsennellä eri kategorioihin pelkän järjestämättömän listan sijaan. Mahdollisia kategorioita voisivat olla esimerkiksi laitteen yhteydet, suojausominaisuudet ja sekä tietoturvapäivitysten ajantasaisuus. Nämä kategoriat voisivat sitten saada omat riskitasoarvonsa.

Riskitason laskemisen perusteena olevat riskipainot joka osatekijälle kaipaisivat myös selkeämpää laskemistapaa ja perustelua käyttäjälle. Painoarvon laskemisen perusteet olisi hyvä myös miettiä tarkemmin nykyisen, ehkä liian yksinkertaistetun muodon korvaamiseksi. Sen tulee kuitenkin olla selkeä ja helposti käyttäjälle perusteltavissa.

Sovellusta käytettäessä tuli myös havainto, että riskitasoluvun pienentämisessä lähelle nollaa oli myös pelillinen ulottuvuus. Sovellusta testattaessa uudella laitteella sen riskitason vähentämisellä oli viihteellistä arvoa. Tietoturva-asioiden opettamista pelillistämisen kautta onkin tutkittu jo aiemmin ja sitä kautta saadut tulokset ovat kannustavia [103, 104]. Yksi tapa kehittää sovelluksesta ja tietoturvan osatekijöistä helpommin lähestyttäviä ja ymmärrettäviä olisi sovelluksen pelillistäminen soveltuvilta osin.

5.3.3. *Riskitason esitysmuoto muille sovelluksille*

Tieto riskitasosta lähetetään toiselle sovellukselle intent-muodossa, jossa riskitaso on Extra-lisätietokentässä kokonaislukuna väliltä 0–100. Tämän toteutustapa on esitetty kuvassa 32. Tämä tapa on varsin yksinkertaistava siihen nähden, mitä tietoa luvun taakse asettuu.

```
private void sendRiskLevelUpdateIntent() {
    updateRiskLevel();

    Intent mIntent = new Intent();
    mIntent.putExtra( name: "risklevel", riskLevel);
    mIntent.setPackage("com.example.app");
    mIntent.setClassName( packageName: "com.example.app",
            className: "com.example.app.ReceiverService");

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        this.startForegroundService(mIntent);
    } else {
        this.startService(mIntent);
    }
    Log.i(TAG,  msg: "Sent intent with data: " + riskLevel);
}
```

Kuva 32. Metodi intentin lähetykseen.

Järkevämpi tapa riskitason välitykseen yhden luvun sijasta olisi lähettää jokaisen riskitason tila erikseen, jolloin toinen sovellus saisi poimia niistä oleelliset ja tehdä valistuneempia ratkaisuja laitteen tietoturvaominaisuuksien muutosten suhteen. Yksi tapa lähettää osatekijöiden tiedot toiselle sovellukselle olisi sisällyttää ne JSON-objektiin, joka sitten lisättäisiin intentin Extra-lisätietokenttään. Tieto voisi olla esimerkiksi allaolevassa muodossa.

{ “riskLevelData” : “
  { "factor" : isDeviceRooted,
    "value" : false },
  { "factor" : isScreenLockEnabled,
    "value" : true },
  { "factor" : areDevelopmentSettingsEnabled,
    "value" : false },
  { "factor" : arePasswordsShown,
    "value" : null }”
}

5.3.4. *Järjestelmätason sovelluksen mahdollisuudet*

Järjestelmätason sovelluksella voitaisiin saada paljon enemmän tietoturvaan liittyvää tietoa laitteesta. Tällainen sovellus vaatii kuitenkin pääkäyttäjätason oikeudet laitteeseen tai mahdollisuuden rakentaa oma järjestelmäkuva, johon sovellus on sisällytettynä. Tämä itse rakennettu kuva pitää sitten ylikirjoittaa laitteeseen sen vanhan järjestelmäkuvan päälle. Tällainen sovellus olisi mahdollista tehdä sen

projektin sisällä, jonka ohessa tämä työ tehtiin. Tämän kehitysidean pohdinta on siten tärkeää työn jatkamisen osalta. Näin tehtävää järjestelmätason sovellusta ei saisi siis asennettua tavalliseen, roottamattomaan laitteeseen esimerkiksi sovelluskaupan kautta. Joidenkin osatekijöiden tarkistusmetodit voivat myös olla laitekohtaisia, jolloin ne eivät toimi kuin tietynmallisissa laitteissa.

Pääsy järjestelmätason oikeuksiin antaisi pääsyn tarkistamaan muun muassa alla mainittuja osatekijöitä. Tavallisilla sovelluksilla ei ole pääsyä näihin tietoihin.

- Muistikortin salauksen käyttö
- SIM-kortin PIN-koodin käyttö
- Välitön lukitus virtapainiketta painamalla
- Näytön lukituksen kompleksisuus (Android 10 -versiota vanhemmat laitteet) [101]
- Ilmoitusten sisällön näkyvyys lukitusnäytöllä (koko sisältö näkyvillä, arkaluonteinen sisältö piilotettu tai ei lainkaan ilmoituksia)
- Smart Lock -lukituksen käyttö
- Paikanna puhelin -ominaisuuden käyttö
- Itse asennetut varmenteen myöntäjät (engl. *certificate authority, CA*)
- Google Play Protect -suojauksen käyttö
- Automaattinen päivän ja ajan päivitys
- Tuntemattomien sovelluslähteiden käytön tarkistus Android 8 -versiosta ylöspäin

Järjestelmätason käyttöoikeuksilla voisi tarkastella tietoturvaa myös paljon laajemmin kuin vain edellä mainittujen osatekijöiden kannalta. Järjestelmätason oikeuksilla voisi tarkastella tarkemmin myös muiden sovelluksien toimintaa, niille myönnettyjä käyttöoikeuksia sekä laitteen järjestelmän tapahtumia ja tilaa esimerkiksi *dumpsys*- ja *systrace*-komentojen avulla. Näiden tarkkailu manuaalisesti olisi liian hankalaa, joten yksi vaihtoehto olisi havaita ja tunnistaa mahdollisia tietoturvapoikkeamia koneoppimisen avulla. Koneoppimisalgoritmia tulisi siis opettaa ensin vahingollisilla sovelluksilla. Sovelluksien ja lokitietojen tarkkailu on kuitenkin niin laaja osa-alue, että niihin perustuva sovellus kannattaisi tehdä pikemmin kokonaan omana projektinaan kuin jatkona nykyiselle.

Järjestelmätason sovellusta kehittäessä tietoturvallinen kehittämistapa ja koodi ovat vielä tärkeämpiä kuin normaalia sovellusta kehittäessä. Järjestelmätason sovellus pitää suojata tarkasti ja katsoa, etteivät sen mahdolliset rajapinnat ja tiedonsiirto muiden sovellusten välillä jätä aukkoa haittasovelluksille niin, että ne saisivat tätä kautta esimerkiksi järjestelmätason oikeuksia.

6. YHTEENVETO

Työn tavoitteena oli kehittää Android-sovellus, joka mittaisi tietoturvan osatekijöitä laitteesta ja laskisi näiden perusteella tietoturvan riskitason. Android-laitteen käyttäjä pystyisi sovelluksen avulla tarkistamaan, onko laitteen tietoturva alttiina ja mihin asioihin siinä voisi kiinnittää huomiota. Muiden sovellusten toiminta ja laitteen lokitietojen tarkkailu rajattiin tämän tarkastelun ulkopuolelle. Tieto mitatusta tietoturvan riskitasosta tuli myös pystyä lähettämään toiselle sovellukselle, joka voi olla esimerkiksi järjestelmätasolla toimiva sovellus. Tämä toinen sovellus saattaisi riskitason kohoamisen perusteella tehdä muutoksia laitteen toimintaan, ehkäisten näin mahdollisia tietoturvauhkia. Kehitetty sovellus tuli olla asennettavissa tavallisena sovelluksena mihin tahansa Android-laitteeseen, jonka käyttöjärjestelmäversio on väliltä 6–9.

Työssä tehtiin yleiskatsaus tietoturvaan ja pureuduttiin tarkemmin siihen, miten tietoturva on otettu huomioon ja toteutettu Androidissa. Lisäksi perehdyttiin tarkemmin laitteista mitattavissa oleviin tietoturvan osatekijöihin ja erityisesti siihen, miksi ne olivat tietoturvan kannalta olennaisia asioita tarkistaa ja korjata. Tällaisia mitattavia osatekijöitä ovat esimerkiksi näytön lukituksen käyttäminen, avoimien wifi-verkkojen käyttäminen ja laitteen tallennustilan salaaminen.

Tehty sovellus täytti sille asetetut tavoitteet. Sovellusta kehitettäessä vastaan tuli rajoituksia joidenkin mitattaviksi aiottujen osatekijöiden osalta, kuten esimerkiksi pääsykoodin kompleksisuuden mittaamisessa. Androidin sovelluksille asetettujen rajoitusten vuoksi näitä kaikkia ei voitu mitata ja aikarajoitusten takia osaa suunnitelluista osatekijöiden mittauksista ei ehditty implementoida sovellukseen tämän työn osana.

Sovelluksen oli tarkoitus olla enemmän tietoturvan mittaamiseen tarkoitetun sovellusidean potentiaalin selvittämistä kuin valmis kaupallinen sovellus. Laitteen mittaama ja ilmoittama riskitaso ei ole kaikenkattava ja esimerkiksi laitteen tietoturvaominaisuuksiin liittyviä osatekijöitä voisi tarkastella enemmän. Tässä työssä tehty sovellus on kuitenkin hyvä pohja jatkokehitykselle ja siihen tulikin lisää ideoita samalla kun sovelluksen tekeminen eteni. Muun muassa riskitason laskeminen ja esittäminen käyttäjille helposti ymmärrettävässä muodossa vaatisi kehittämistä, samoin kuin riskitasotiedon liian yksinkertaistava muoto toiselle sovellukselle lähetettäessä. Sovelluksen käyttäjäkokemusta voisi parantaa mahdollisesti myös sen pelillistämisen kautta.

Kehitetty sovellus toimii tavallisena, ei-järjestelmätason sovelluksena. Projektin sisällä, jonka ohessa tämä työ tehtiin, olisi mahdollista tehdä myös järjestelmätason sovellus Android-laitteelle, joka saisi siis laajemmat käyttöoikeudet kuin tavallinen sovellus. Se pystyisi näin ollen tarkistamaan tietoturvan osatekijöitä, joihin tavallisella sovelluksilla ei ole pääsyä. Tietoturvan riskitasoa voisi myös mitata tutkimalla muita sovelluksia, niiden toimintaa ja laitteen lokitietoja, mutta näiden osa-alueiden laajuuden takia niiden tutkiminen kannattaisi eriyttää omaksi projektikseen.

7. LÄHTEET

[1] StatCounter. (2020). Mobile & tablet operating system market share worldwide. URL: https://gs.statcounter.com/os-market-share/mobile-tablet/worldwide/#monthly-201208-202001. Haettu 29.1.2020.

[2] Android Open Source Project. (2019). Application sandbox. URL: https://source.android.com/security/app-sandbox. Haettu 10.10.2019.

[3] StatCounter. (2020). Desktop vs Mobile vs Tablet Market Share Worldwide. URL: https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/worldwide/#monthly-200901-202001. Haettu 30.1.2020.

[4] Khokhlov, I., & Reznik, L. (2018). Android system security evaluation. In 2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC) (pp. 1-2). IEEE. URL: https://www.researchgate.net/publication/323864674_Android_system_security_evaluation. DOI: https://dx.doi.org/10.1109/CCNC.2018.8319325.

[5] Google Play. DTEK by BlackBerry. URL: https://play.google.com/store/apps/details?id=com.blackberry.privacydashboard. Haettu 9.9.2019.

[6] ISO/IEC. (2018). ISO/IEC 27000:2018(en) Information technology - security techniques - information security management systems - overview and vocabulary. URL: https://www.iso.org/obp/ui/#iso:std:iso-iec:27000:ed-5:v1:en. Haettu 3.12.2019.

[7] Andress, J. (2011). The basics of information security - understanding the fundamentals of InfoSec in theory and practice, Second edition.

[8] SELinux Project. (2009). FAQ. URL: https://selinuxproject.org/page/FAQ. Haettu 5.12.2019.

[9] Android Open Source Project. Security-Enhanced Linux in Android. URL: https://source.android.com/security/selinux/. Haettu 5.12.2019.

[10] De Laat, C. T. A. M., Gross, G., Gommans, L., Vollbrecht, J., & Spence, D. (2000). Generic AAA architecture. RFC 2903. URL: https://www.rfc-editor.org/rfc/rfc2903.html. Haettu 16.1.2020.

[11] Aboba, B., Calhoun, P. R., Glass, S. M., Hiller, T., McCann, P. J., Shiino, H., ... Patil, B. (2000). Criteria for evaluating AAA protocols for network access, RFC 2989. URL: https://tools.ietf.org/html/rfc2989. Haettu 16.1.2020.

[12] Kohnfelder, L. & Garg, P., Microsoft Corporation. (1999). The threats to our products. URL: https://adam.shostack.org/microsoft/The-Threats-To-Our-Products.docx. Haettu 5.12.2019.

[13] Howard, M., & LeBlanc, D. (2003). Writing secure code. Pearson Education.

[14] Open Web Application Security Project. (2019). OWASP risk rating methodology. URL: https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology. Haettu 14.11.2019.

[15] Mayrhofer, R., Stoep, J. V., Brubaker, C. & Kralevich, N. (2019). The Android platform security model. arXiv preprint arXiv:1904.05572. URL: https://arxiv.org/abs/1904.05572.

[16] Android Open Source Project. Sign your app. URL: https://developer.android.com/studio/publish/app-signing. Haettu 11.12.2019.

[17] Android Open Source Project. Verified boot. URL: https://source.android.com/security/verifiedboot. Haettu 12.12.2019.

[18] Android Open Source Project. System security best practices. URL: https://source.android.com/security/best-practices/system. Haettu 12.12.2019.

[19] Android Open Source Project. App manifest overview. URL: https://developer.android.com/guide/topics/manifest/manifest-intro. Haettu 22.8.2019.

[20] Android Open Source Project. Hardware-backed keystore. URL: https://source.android.com/security/keystore. Haettu 30.12.2019.

[21] Android Open Source Project. CTS test for Secure Element. URL: https://source.android.com/compatibility/cts/secure-element. Haettu 31.12.2019.

[22] Android Open Source Project. Gatekeeper. URL: https://source.android.com/security/authentication/gatekeeper. Haettu 31.12.2019.

[23] Android Open Source Project. Full-disk encryption. URL: https://source.android.com/security/encryption/full-disk.html. Haettu 10.9.2019.

[24] Android Open Source Project. (2019). File-based encryption. URL: https://source.android.com/security/encryption/file-based.html. Haettu 10.9.2019.

[25] Android Open Source Project. Metadata encryption. URL: https://source.android.com/security/encryption/metadata. Haettu 14.1.2020.

[26] Android Open Source Project. Security with HTTPS and SSL. URL: https://developer.android.com/training/articles/security-ssl. Haettu 31.12.2019.

[27] Android Open Source Project. App manifest file, application elements. URL: https://developer.android.com/guide/topics/manifest/application-element#usesCleartextTraffic. Haettu 31.12.2019.

[28] Kralevich, N. (2016). The Art of Defense. How vulnerabilities help shape security features and mitigations in Android. URL: https://www.blackhat.com/docs/us-16/materials/us-16-Kralevich-The-Art-Of-Defense-How-Vulnerabilities-Help-Shape-Security-Features-And-Mitigations-In-Android.pdf. Haettu 2.1.2020.

[29] Austin, D. & Stoep, J. V., Android Developers Blog. (2016). Hardening the media stack. URL: https://android-developers.googleblog.com/2016/05/hardening-media-stack.html. Haettu 2.1.2020.

[30] Android Open Source Project. Control flow integrity. URL: https://source.android.com/devices/tech/debug/cfi. Haettu 2.1.2020.

[31] Tolvanen, S., Android Developers Blog. (2016). Strictly enforced verified boot with error correction. URL: https://android-developers.googleblog.com/2016/07/strictly-enforced-verified-boot-with.html. Haettu 4.1.2020.

[32] Android Open Source Project. Verified boot: verifying boot. URL: https://source.android.com/security/verifiedboot/verified-boot#rollback-protection. Haettu 3.1.2020.

[33] Android Open Source Project. Verifying hardware-backed key pairs with key attestation. URL: https://developer.android.com/training/articles/security-key-attestation. Haettu 4.1.2020.

[34] Android Open Source Project. Application signing. URL: https://source.android.com/security/apksigning/. Haettu 4.1.2020.

[35] Malchev, I., Android Developers Blog. (2017). Here comes Treble: a modular base for Android. URL: https://android-developers.googleblog.com/2017/05/here-comes-treble-modular-base-for.html. Haettu 7.1.2020.

[36] Mohamed, I., & Patel, D. (2015). Android vs iOS security: a comparative study. In 2015 12th International Conference on Information Technology-New Generations (pp. 725-730). IEEE. URL: https://ieeexplore.ieee.org/abstract/document/7113562. DOI: https://doi.org/10.1109/ITNG.2015.123.

[37] Google. (2019). Set screen lock on an Android device. URL: https://support.google.com/android/answer/9079129?hl=en. Haettu 2.5.2019.

[38] The Linux Information Project. (2007). Root definition. URL: http://www.linfo.org/root.html. Haettu 8.10.2019.

[39] Drake, J. J., Lanier, Z., Mulliner, C., Fora, P. O., Ridley, S. A. & Wicherski, G. (2014). Android hacker's handbook. John Wiley & Sons.

[40] Unuchek, R., AO Kaspersky Lab. (2017). Rooting your Android: advantages, disadvantages, and snags. URL: https://www.kaspersky.com/blog/android-root-faq/17135/. Haettu 10.10.2019.

[41] Titanium Track LLC. About Titanium Backup. URL: https://www.titaniumtrack.com/titanium-backup.html. Haettu 11.10.2019.

[42] Coding Code Mobile Technology. (2017). SuperSU. URL: http://www.supersu.com. Haettu 11.10.2019.

[43] Snyder, J., Samsung. (2019). What are the security risks of rooting your smartphone? URL: https://insights.samsung.com/2019/05/29/what-are-the-security-risks-of-rooting-your-smartphone/. Haettu 11.10.2019.

[44] The LineageOS Project. (2019). LineageOS Android distribution. URL: https://lineageos.org/. Haettu 11.10.2019.

[45] Sun, S. T., Cuadros, A. & Beznosov, K. (2015). Android rooting: methods, detection, and evasion. In Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices (pp. 3-14). ACM. URL: http://lersse-dl.ece.ubc.ca/record/310/files/p3.pdf.

[46] Piana, C. & Šuklje, M., Free Software Foundation Europe. (2012). Does rooting your device (e.g. an Android phone) and replacing its operating system with something else void your statutory warranty, if you are a consumer? URL: https://fsfe.org/freesoftware/legal/flashingdevices.en.html. Haettu 14.10.2019

[47] Euroopan Unioni. (1999). Euroopan parlamentin ja neuvoston direktiivi 1999/44/EY. URL: https://eur-lex.europa.eu/legal-content/FI/TXT/HTML/?uri=CELEX:31999L0044. Haettu 14.10.2019.

[48] Euroopan Unioni. Tuotetakuu. URL: https://europa.eu/youreurope/business/dealing-with-customers/consumer-contracts-guarantees/consumer-guarantees/index_fi.htm. Haettu 14.10.2019.

[49] Center for Internet Security. (2019). CIS Google Android benchmarks (2019). URL: https://www.cisecurity.org/benchmark/google_android/. Haettu 28.1.2019.

[50] Android Open Source Project. Android debug bridge. URL: https://developer.android.com/studio/command-line/adb. Haettu 24.7.2019.

[51] Android Open Source Project. Logcat command-line tool. URL: https://developer.android.com/studio/command-line/logcat. Haettu 25.7.2019.

[52] Android Open Source Project. Getevent-tool. URL: https://source.android.com/devices/input/getevent. Haettu 25.7.2019.

[53] Hwang, S., Lee, S., Kim, Y. & Ryu, S. (2015). Bittersweet ADB: attacks and defenses. In Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security (ASIA CCS '15). Singapore, Singaporen tasavalta, 579-584. URL: https://syssec.kaist.ac.kr/pub/2015/hwang_asiaccs2015.pdf. DOI: https://doi.org/10.1145/2714576.2714638.

[54] Bluetooth SIG. (2019). Bluetooth core specification v5.1. URL: https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=457080. Haettu 7.8.2019.

[55] IEEE. (2019). MAC address block large (MA-L). URL: https://standards.ieee.org/products-services/regauth/oui/index.html. Haettu 7.8.2019.

[56] Sofer, N. (2016). NirSoft BluetoothLogView v1.12. URL: http://www.nirsoft.net/utils/bluetooth_log_view.html. Haettu 5.8.2019.

[57] Trifinite group. BlueSnarf. URL: https://trifinite.org/trifinite_stuff_bluesnarf.html. Haettu 9.8.2019.

[58] Laurie, A. & Laurie, B., A.L. Digital Ltd. (2003). Serious flaws in bluetooth security lead to disclosure of personal data. URL: https://www.bluestumbler.org/. Haettu 9.8.2019.

[59] Bahr, J., Batra, M., Chen, L., Holtmann, M., Padgette, J., Scarfone K. & Smithbey, R. (2017). Guide to Bluetooth security. NIST Special Publication. URL: https://www.nist.gov/publications/guide-bluetooth-security-1. DOI: https://doi.org/10.6028/NIST.SP.800-121r2.

[60] Trifinite group. BlueBug. URL: https://trifinite.org/trifinite_stuff_bluebug.html. Haettu 13.8.2019.

[61] Chen, W. L. & Wu, Q. (2010). A proof of MITM vulnerability in public WLANs guarded by captive portal. Proceedings of the Asia-Pacific Advanced Network, 30, 66-70. URL: http://journals.sfu.ca/apan/index.php/apan/article/download/80/pdf_36.

[62] Green, I. (2005). DNS spoofing by the man in the middle. URL: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.180.5419.

[63] Aircrack-ng. (2018). Airodump-ng. URL: https://www.aircrack-ng.org/doku.php?id=airodump-ng. Haettu 16.8.2019.

[64] Europol. Risks of using public Wi-Fi. URL: https://www.europol.europa.eu/activities-services/public-awareness-and-prevention-guides/risks-of-using-public-wi-fi. Haettu 19.8.2019.

[65] Hoffman, C., How-To Geek, LLC. (2018). What is HTTPS, and why should I care? URL: https://www.howtogeek.com/181767/htg-explains-what-is-https-and-why-should-i-care/. Haettu 19.8.2019.

[66] Clark, J. & Van Oorschot, P. C. (2013). SoK: SSL and HTTPS: revisiting past challenges and evaluating certificate trust model enhancements. In 2013 IEEE Symposium on Security and Privacy (pp. 511-525). IEEE. URL: https://ieeexplore.ieee.org/abstract/document/6547130. DOI: https://doi.org/10.1109/SP.2013.41.

[67] Google. About Google Maps. URL: https://www.google.com/maps/about/. Haettu 20.8.2019.

[68] WhatsApp Inc. (2019). FAQ - Using live location. URL: https://faq.whatsapp.com/en/android/26000049/. Haettu 20.8.2019.

[69] Sports Tracking Technologies. (2019). Sport Tracker. URL: https://www.sports-tracker.com/. Haettu 20.8.2019.

[70] c:geo. (2019). FAQ. URL: https://www.cgeo.org/faq. Haettu 20.8.2019.

[71] Niantic, Inc., Pokémon, Nintendo, Creatures Inc., Game Freak Inc. (2019). Frequently asked questions. URL: https://pokemongolive.com/en/faq/. Haettu 20.8.2019.

[72] Google. (2019). Location data. URL: https://developers.google.com/maps/documentation/android-sdk/location. Haettu 22.8.2019.

[73] Android Open Source Project. Permissions overview. URL: https://developer.android.com/guide/topics/permissions/overview. Haettu 22.8.2019.

[74] EU. (2016). Euroopan parlamentin ja neuvoston asetus (EU) 2016/679. URL: https://eur-lex.europa.eu/legal-content/FI/TXT/HTML/?uri=CELEX:32016R0679. Haettu 27.8.2019.

[75] National Institute of Standards and Technology. (2001). Announcing the Advanced Encryption Strandard (AES). URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf. DOI: https://doi.org/10.6028/NIST. FIPS.197.

[76] Ferris, S., Leech, D. & Scott, J. (2018). The economic impacts of the advanced encryption standard, 1996-2017. URL: https://nvlpubs.nist.gov/nistpubs/gcr/2018/NIST.GCR.18-017.pdf. DOI: https://doi.org/10.6028/NIST.GCR.18-017.

[77] Seagate Technology. (2008). 128-bit versus 256-bit AES encryption. URL: http://www.axantum.com/axcrypt/etc/seagate128vs256.pdf. Haettu 1.10.2019.

[78] Arora, M., EE Times. (2012). How secure is AES against brute force attacks? URL: https://www.eetimes.com/document.asp?doc_id=1279619. Haettu 1.10.2019.

[79] Stallings, W. (2017). Cryptography and network security: principles and practice (7th edition). Pearson Education Limited, Englanti.

[80] Bennett, C. L., Larson, D., Weiland, J. L., Jarosik, N., Hinshaw, G., Odegard, N., ... & Komatsu, E. (2013). Nine-year Wilkinson Microwave Anisotropy Probe (WMAP) observations: final maps and results. The Astrophysical Journal Supplement Series, 208(2), 20. URL: https://arxiv.org/pdf/1212.5225.pdf. DOI: https://doi.org/10.1088/0067-0049/208/2/20.

[81] Bogdanov, A., Khovratovich, D., & Rechberger, C. (2011). Biclique cryptanalysis of the full AES. URL: https://eprint.iacr.org/2011/449.pdf.

[82] Standaert, F. X. (2010). Introduction to side-channel attacks. In Secure Integrated Circuits and Systems (pp. 27-42). Springer, Boston, MA. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1019.7718&rep=rep1&type=pdf.

[83] Ashokkumar, C., Giri, R. P., & Menezes, B. (2016). Highly efficient algorithms for AES key retrieval in cache access attacks. URL: http://www.academia.edu/download/45039893/Euro_Final_Final.pdf.

[84] Etemadieh, A., Heres, C. & Hoang, K., Exploitee.rs (2017). Hacking hardware with a $10 SD card reader. URL: https://www.blackhat.com/docs/us-17/wednesday/us-17-Etemadieh-Hacking-Hardware-With-A-$10-SD-Card-Reader-wp.pdf.

[85] Khramova, M. & Martinez, S., Blancco Technology Group. (2019). Analysis of data remanence after factory reset, and sophisticated attacks on memory chips. URL: http://www.blancco.com/wp-content/uploads/2019/04/201811-SSpaper-DataRemanence.pdf.

[86] Android Open Source Project. Android compatibility program overview. URL: https://source.android.com/compatibility/overview#licensing-gms. Haettu 10.1.2020.

[87] Google. Google Play Protect. URL: https://www.android.com/play-protect/. Haettu 8.1.2020.

[88] F-Droid Limited and Contributors. (2020). About F-Droid. URL: https://f-droid.org/en/about/. Haettu 17.1.2020.

[89] Android Open Source Project. Android 8.0 behavior changes. URL: https://developer.android.com/about/versions/oreo/android-8.0-changes. Haettu 8.1.2020.

[90] Newman, L. H., Wired. (2017). How malware keeps sneaking past Google Play's defenses. URL: https://www.wired.com/story/google-play-store-malware/. Haettu 13.1.2020.

[91] Tee, M. Y. & Zhang, M., Symantec. (2019). More hidden app malware found on Google Play with over 2.1 million downloads. URL: https://www.symantec.com/blogs/threat-intelligence/hidden-adware-google-play. Haettu 13.1.2020.

[92] Kleidermacher, D., Google. (2019). The App Defense Alliance: bringing the security industry together to fight bad apps. URL: https://security.googleblog.com/2019/11/the-app-defense-alliance-bringing.html. Haettu 13.1.2020.

[93] Google. Choose when your Android device can stay unlocked. URL: https://support.google.com/android/answer/9075927?hl=en. Haettu 21.1.2020.

[94] Android Open Source Project. Distribution dashboard. URL: https://developer.android.com/about/dashboards. Haettu 29.10.2019.

[95] Android Open Source Project. Background execution limits. URL: https://developer.android.com/about/versions/oreo/background.html. Haettu 4.11.2019.

[96] Android Open Source Project. (2019). Intents and intent filters. URL: https://developer.android.com/guide/components/intents-filters. Haettu 21.2.2020.

[97] BlackBerry. (2016). DTEK50 by BlackBerry - DTEK by BlackBerry: Official How To Demo [video]. URL: https://www.youtube.com/watch?v=k3YKbNmyTj8.

[98] Android Open Source Project. (2019). SafetyNet Attestation API. URL: https://developer.android.com/training/safetynet/attestation. Haettu 4.3.2020.

[99] Android Open Source Project. (2019). SafetyNet Verify Apps API. URL: https://developer.android.com/training/safetynet/verify-apps. Haettu 4.3.2020.

[100] Google. Help protect against harmful apps with Google Play Protect. URL: https://support.google.com/accounts/answer/2812853?hl=en. Haettu 4.3.2020.

[101] Android Open Source Project. What's new for enterprise in Android 10. URL: https://developer.android.com/work/versions/android-10#screen_lock_quality_check. Haettu 25.1.2020.

[102] Nguyen-Vu, L., Chau, N. T., Kang, S., & Jung, S. (2017). Android rooting: an arms race between evasion and detection. URL: https://pdfs.semanticscholar.org/d38c/cc4e80bf78b435da3916b9ecd118561ac472.pdf.

[103] Thornton, D., & Francia, G. (2014). Gamification of information systems and security training: Issues and case studies. Information Security Education Journal, 1(1), 16-24. URL: http://www.dline.info/isej/fulltext/v1n1/3.pdf.

[104] Banfield, J., & Wilkerson, B. (2014). Increasing student intrinsic motivation and self-efficacy through gamification pedagogy. Contemporary Issues in Education Research (CIER), 7(4), 291-298. URL: https://www.clutejournals.com/index.php/CIER/article/download/8843/8809.